# HybrIDS: EMBEDDABLE HYBRID INTRUSION DETECTION SYSTEM

By

Adrian Peter Lauf

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

December, 2007

Nashville, Tennessee

Approved:

Professor William H. Robinson

Professor Richard A. Peters

TABLE OF CONTENTS

TABLE OF FIGURES

## LIST OF ACRONYMS

| | |
|---|---|
| ARM9 | ARM embedded processor version 4 variant 920T |
| AES | Advanced Encryption Standard |
| CCIDS | Cross-Correlative Intrusion Detection System |
| (3)DES | Digital Encryption Standard |
| DCC | Data Collection Cycle |
| DPC | Data Processing Cycle |
| HybrIDS | Hybrid Intrusion Detection System |
| IDS | Intrusion Detection System |
| MDS | Maxima Detection System |
| WEP | Wired Equivalency Protection |

## ACKNOWLEDGMENTS

It is with sincere gratitude that I wish to thank my advisor, Dr. William H. Robinson, for his continued support and guidance during the development of the IDS systems and accompanying documents. Without his help, this work would not exist. I also thank Dr. Richard A. Peters for his guidance in algorithmic development and methodology – key to developing the cross-correlative IDS section of this work. To my parents, Drs. Peter K. and Norma C. Adragna Lauf my gratitude for their constant care and their professional revision of my work. To my girlfriend Carrie, I give my thanks for her continued support and help during the development of this work. Finally, I wish to thank TRUST (The Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244) Cisco, British Telecom, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia and United Technologies.

# ABSTRACT

In order to provide preventative security to a homogeneous device network, techniques in addition to static encryption must be implemented to assure network integrity by identifying possible deviant nodes within the collective. This thesis proposes a set of algorithms and techniques for an intrusion detection system, which when combined, provide a two-stage approach that seeks to reduce or eliminate training period requirements, while providing multiple anomaly detection and a degree of self tuning. By utilizing a high level of behavioral abstraction, these intrusion detection techniques can be applied to a broad range of devices, network implementations, and scenarios. Each device node is supplied with an embedded intrusion detection system which allows it to monitor inter-device requests, enabling machine learning techniques for purposes of deviant node analysis. The two principal methods, a maxima detection scheme, and a cross-correlative detection scheme, are combined to create a two-phase detection scheme that can successfully determine deviant node pervasion percentages of up to 22% within the homogeneous device network.

# CHAPTER I

# INTRODUCTION

When analyzing the broad range and applications utilizing concepts and designs for embedded systems, the importance of inter-device communications becomes extremely clear: communication and collaboration among devices serve as the backbone for system productivity in a highly networked environment. For this reason, there are implications in the methods, protocols and design considerations whereby communication is established, and arguably just as important, protected. Thus, information security, in its many forms, provides a needed level of protection to communications protocols present within the scope of interconnected devices. For the most part, information security relies on preventive methods to protect content, using static techniques such as obfuscation of source, encryption of data (satisfying the need for confidentiality), and source integrity by means such as digital signatures. As a first line of defense, static security methods are steadfast and proven methods of protecting data; The RSA[1-4] encryption standard, 3DES[4, 5], and AES[6, 7] all remain, at the time of this writing, effective and secure.

Perhaps the biggest obstacle to providing accurate information security using static methods is that a point of trust is always required; at some point, there must exist a trusted resource relationship, whether it is in the transmission of a public key[1], or a

[1] RSA encryption relies on the difficulty of factoring near-prime numbers; because a common public key combined with a strong private key yields a nearly un-factorable challenge, RSA is considered safe.

shared secret. Because of this trusted relationship, there always exists the possibility that despite the integrity and strength of the algorithm being used, the static method can be defeated, however unlikely the scenario may be. For instance, WEP[2] [8, 9] encryption commonly used to secure wireless networks relies on a solid, trustworthy encryption method that is traditionally known to be safe. However, the particular implementation of the algorithm within the scope of the wireless network protocol is flawed [10], allowing the security protocol to be cracked within minutes, given enough data, time and resources.

Because of this point-of-trust issue, the most disturbing problem with static protection methods is that once broken, they are no longer able to ensure the trustworthiness of the system to which they are applied. Even more problematic is providing any sort of detection that the security implementation has been broken in the first place. Although temporal breach protection solutions exist, such as encryption/decryption key rotations and replacement algorithms, it would not be unforeseen that broken once, the security scheme can be broken again. To ameliorate the damage from such breaches, measures containing dynamic approaches to security must be considered. In particular, intrusion detection provides means by which anomalies in general system behaviors can be analyzed and graded on the threat they pose.

Such a dynamic system would have the capabilities of securing a wide variety of applications, from general networked computing to specialized, applied embedded

---

Recently, a commonly-held public key used for 1024-bit RSA encryption was defeated (indicating that the private key could be factored without prior knowledge).

[2] Wired Equivalency Protection

devices sharing a network connection. An Intrusion Detection System[3] may operate in a variety of methods, such as performing individual packet-level analysis of incoming and outgoing network traffic. Indeed, IDSs typically use packet monitoring [1, 7, 11-23] with the aid of a probabilistic model to determine whether or not a network is the subject of an intrusion. Primarily effective in determining intrusions in protocol-specific environments where the range of input data is unbounded[4], traffic-based IDSs are not as applicable to the embedded systems paradigm. In this case, an applied computational scenario has been established and localized to a set of known behaviors and parameters.

This work describes an embeddable IDS that utilizes the known aspects of an applied system's communications network, whether present in an autonomous vehicle collective or mobile sensor device network, to form the basis for accurate intrusion detection in a low-power, high-level context. Two principal methods of intrusion detection are proposed and analyzed, and ultimately hybridized to create a resultant IDS that delivers the capability of detecting multiple intrusions along with low-resource utilization and a desirable level of system accuracy.

In order to provide a desirable level of intrusion detection, and of equal importance, resistance to attempts to defeat the IDSs functionality and accuracy, the IDS mechanisms presented in this work seek to thwart intrusion by making an attack much more difficult to plan and execute. Of course, no IDS is perfect unless the operational context is static and exactly prescribed before runtime, a case scenario that is not considered as it is impractical to consider for real-world applications. Therefore, the IDS

[3] This document shall refer to an Intrusion Detection System with the acronym IDS
[4] Considered unbounded as the data contained in packets is not analyzed; the content is not considered

mechanisms presented here, and summarized by a hybridized approach, offer resistance in various forms:

1. **Requiring the attacker to have intimate knowledge of the system it is attacking:** In order to escape detection by an IDS, an attacker would need to know all operational details with a high level of precision and determinism. This is often information that is not accessible or available, and thus lack of intimate knowledge makes planning difficult for an attacker.
2. **Extending the attack time window to impractical lengths:** Because the IDSs use machine learning and statistical analysis methods over time, the attacker would need to expand its mission timeframe so severely that the attack may not be successful.
3. **Requiring timed injections:** An attacker must be able to time its injections and pad its behavior with "normal" system behaviors so that its behavior does not trip thresholds and monitoring techniques that rely on statistical inference and temporal study. To do this, the attacker would need to master the "intimate knowledge" point in this list, further complicating an attack.

The points listed above present reasons by which a successful IDS can thwart an attacker – by making the possible attack very difficult to plan, time, and implement. The methods seen in the next chapters help to mitigate attack potential by: (1) complicating a potential attack, (2) making it unfeasible, and (3) reducing the possibility of network compromise.

## Organization

This thesis work will be organized as follows:

- Chapter Two: Necessary concepts for understanding IDS methodology, constructs, and an example scenario
- Chapter Three: Description of the first IDS method, called Maxima Detection
- Chapter Four: Description of the second IDS method, called the Cross-Correlative Intrusion Detection System
- Chapter Five: A description of a hybridized IDS, called HybrIDS
- Chapter Six: A conclusion detailing the overall implications of the methodologies

# CHAPTER II

# CONCEPT PRIMITIVES AND SCENARIO

To better illustrate the functional aspects of the IDSs that will be described in the next few chapters, it is worthwhile to introduce an operational scenario to examine the application and contextual aspects of the IDS. In particular, this document will focus on the benefits provided to an ad-hoc network comprised of homogeneous networked nodes. Such a collective may be defined as a group of autonomous aircraft, ground vehicles, networked media players capable of sharing and transmitting data, joint attack smart munitions (such as the U.S. Military's JDAM – Joint Directed Attack Munitions [24]), or any other configuration of networked nodes that comply with the ad-hoc, homogeneous requirement.

## Scenario: ADS-B

The case scenario presented here involves a modified version of the Automatic Dependent Surveillance – Broadcast (ADS-B) system used to provide flight status information and collision avoidance to a network of interconnected aircraft and ground-based receiver stations. For this thesis, the focus will only be on inter-aircraft communications.

To gain an understanding of how an IDS might be integrated into such a broadcast mechanism, it is useful to characterize the existing system. ADS-B broadcast messages contain five unique, aircraft-centric data points and/or vectors, specifying:

- GPS Position information
- Altitude of the aircraft
- Rate of climb
- Velocity vector
- Aircraft ID tag

These data points are broadcast typically at intervals of 2 Hz [25-28], and are received by any nearby aircraft. Software implemented on various other modules aboard the aircraft is then responsible for decoding the broadcast stream and performing decisions according to the information presented.

## Changes to the ADS-B Model

Because IDSs are typically implemented in scenarios where bidirectional communications is required, the current ADS-B specification is therefore not a proper application of IDS technology. Because of this, the ADS-B specification has been adapted to include the need for inter-aircraft requests, and two specific directives for theoretical use in autonomous aircraft missions were added. The modifications are as follows:

- GPS Position information request

- Altitude of the aircraft request
- Rate of climb request
- Velocity vector request
- Mission update request
- Redirection request
- Mission start request
- Mission end request
- Emergency/evasive action request
- Priority/dominance leader/follower change request

The reader will notice that the "ID Tag" item has been removed from the previous specification. It is assumed therefore that in performing bidirectional communications with other aircraft nodes, the notion of the need for a specific aircraft identifier is handled by the communications protocol itself, and thus abstracted away from the purposes of this research.

Six additional functions have been added to the ADS-B system specification, along with the requirement that instead of a non-directed broadcast, each connected aircraft node makes specific requests of other devices according to the newly-proposed items listed above. The first new addition, that of a mission update request, simply is a query to other connected aircraft to supply the requester with an updated profile of its mission information. This allows for dissemination of group policy and provides an updated group dynamics model to each node as time progresses. For instance, should one aircraft identify a hazardous condition that is not immediately apparent to the rest of the collective, subsequent update requests may provide a warning to the rest of the group as the update request is propagated from device to device. The second action is a simple direction change request. This may have several purposes, including collision avoidance should sensor data to each of the nodes become unavailable, compromised, or obscured. Other actions include emergency evasive actions, mission start/end changes, and the last request, to change ordering or dominance in a series of aircraft, allowing for a change in designated roles from one aircraft to another.

This updated system model now represents a small-scale control protocol for a network of autonomous aircraft. The aim of this thesis is not to explore this example system; for reference purposes, it is simply stating a scenario to which the IDS may be applied.

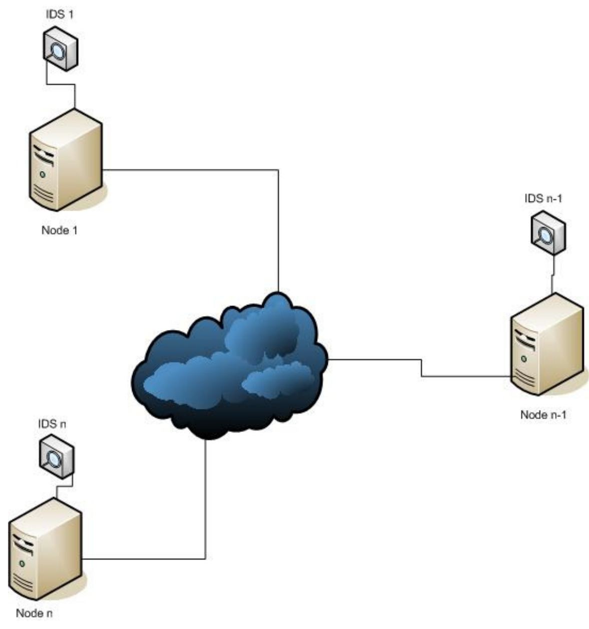


Figure 1 - Example of one IDS per node

## System Integration

As stated in the introductory material, an IDS employing traffic analysis at a single point in a device network is neither scalable (i.e., resists performance degradation as the number of devices increases) nor applicable to an ad-hoc network setup, especially one consisting of power-restricted devices where strenuous computational power requirements yield a major disadvantage in implementation. To combat this issue, all of the IDS systems and strategies discussed in this document will circumvent a single point of analysis by allowing each connected device node to run a specific hardware/software implementation of the IDS mechanism, as seen in Figure 1. By using a parallel approach, we can enable multiple-agent feedback, if required (though not discussed as a solution in this document) to provide data to a variety of intrusion control methods (separate from detection in that it is a solution to a detection) such as a reputation system[5].

Parallelism increases system scalability by removing the burden of analysis from one machine monitoring the entire collective, to multiple devices monitoring only their relevant intercommunications. For instance, in a traffic-based IDS, a collective of eight nodes requires that the IDS monitors all eight nodes. In contrast, in a parallel IDS strategy, let us assume that of those eight nodes, nodes A, B and C communicate. In this case, the implemented IDS models, with reference to the IDS onboard node A, will only need to monitor communications with nodes B and C, assuming that no other communications occur. This brings up the important point that none of the methods outlined in the next few sections self-monitor behavior – this would be more or less redundant when considering a significantly large set of communicating nodes. Self-referenced IDS mechanisms therefore will not be addressed in this document.

[5] Reputation Systems will be discussed shortly

## Reputation Systems

Buchegger and Le Boudec [29] introduce and detail different methods used to form the basis for reputation systems, applicable to the ad-hoc network scenario. Their work focuses primarily on developing a system of node-based reputations for determining optimal and safe strategies for routing data among the nodes. In their work, they show a variety of different trust-based reputation-building mechanisms by which nodes that previously have not interacted with each other can determine whether or not the nodes are trustworthy based on prior accumulated information about each respective node.

Central to their work is the propagation of reputation information among nodes which forms the basis for the group-wide consensus about the trustworthiness of the interconnected nodes. The IDS mechanisms detailed in this thesis focus not on trust propagation and group decision making. Rather, the focus is on detection of an intrusion based on the observations of a single node with reference to the collective in such a way that for N nodes in the network there exist N different system state observations. For this reason, the work presented here is not adherent to the concepts of a reputation system.

## Operational Cycle Division and Scalability

When considering performance issues related to embedded device networks, scalability becomes of paramount importance in the determinacy of response time. To this end, the methods proposed in this document utilize mechanisms to minimize computational overhead and allow for expanded scalability through the implementation of a division between data acquisition cycles of the various IDSs, and data processing and analysis cycles. A cycle[6] is defined as a "run" of the IDS during which either data is collected or analysis is performed.

Data Collection Cycles[7] are dedicated to handling input requests from connected agent nodes. These cycles are of low computational intensity, as they simply map received requests to a predefined structure maintaining a history of external requests. In the case of the IDSs mentioned here, this structure will be referred to as the Agent History Table, which contains all requests received during the IDS runtime. A data collection cycle is run for every input data  point, since the update process is lightweight, using only an increment operation and a node information update request. The vast majority of all cycles performed by the IDS fall under the DCC category. This allows for stabilization of input data patterns for purposes of statistical analysis.

In contrast to a DCC, a Data Processing Cycle[8] is a CPU-intensive IDS run that performs the analysis required to identify deviant agents from the node collective. Performing a meaningful analysis is dependent on the data collected. This has two ramifications; First, it means that power and computational resources can be saved by not performing analysis cycles before sufficient data input has been received. Second, modifying the execution point of a DPC enables system flexibility by allowing a statistically significant change in observed behavior to occur before performing a new analysis on the received request inputs. Utilizing the same data structure updated by the

[6] Cycle and Iteration will be used interchangeably
[7] Abbreviated as DCC, or DC if using the word "cycle" verbosely
[8] Abbreviated DPC or DP if using the word "cycle" verbosely

DCC, the DPC portion of the IDSs determines deviant nodes based on the IDS methods described in the following chapters. Typically, the DPC is run based on the number of inputs received and the number of nodes in the collective. For all three IDS scenarios, excluding the initial exploratory maxima detection system implementation, the DPC is run when a counter measuring the number of DCCs exceeds a value computed by the product of the number of known, locally connected nodes and the number of possible discrete system behaviors.

The performance aspects, accuracy, and power/overhead concerns are directly affected by the DCC/DPC execution ratios as the IDS runtime progresses. More DPC executions cause more computational overhead, but are necessary for any data analysis to occur. When properly spaced with enough DCC executions, **the IDS's performance can be shaped to scale linearly, or even negligibly as the size of the networked node cluster is linearly increased.**

# CHAPTER III

# MAXIMA DETECTION

This section of the document will explain in detail the single-anomaly-detection mechanism mentioned in the introduction, and hereby referred to as a Maxima Detection System[9] [30]. The purpose of this detection algorithm is to provide an accurate mechanism for detecting single anomalies within the context of the embedded systems platform, while maintaining a simple and lightweight execution profile. Within the scope of the Hybrid IDS (as seen in Chapter Five), the MDS allows for identification of either one or zero suspicious nodes for calibrating the sensitivity of the Cross-Correlative Intrusion Detection System.[10] Therefore, the MDS is designed primarily as a first-defense and calibration stage for CCIDS to remove the large number of false positives inherent to that approach.

MDS relies on the creation and updating of probability density functions that approximate the observed behavior between nodes interacting with a specific host node. To simplify a behavior-based model for purposes of creating an experimental version of the MDS, behaviors were categorized statistically and represented by integral data values, one per integer, creating an enumerated list of actions and methods. For instance, in the scenario of a series of networked autonomous aircraft, a request for position data might be assigned logically to integer value ‘1’, a request for attitude data might map to a value of ‘2’, and so on. Each of the behaviors is generated according to a probability density

[9] Abbreviated as MDS
[10] Abbreviated as CCIDS

function (PDF) attributed to the frequency of that behavior's occurrence in an actual embedded, real-time system. All probabilities add to 1 to completely represent the possible behavior space of the system. Figure 2 demonstrates this concept within the scope of a system containing nine separate behaviors. This capability allows the IDS to move beyond the scope of a system-specific implementation, abstracting operations at one of the highest possible levels, (level 1 is used in this paper) as seen in Figure 3.

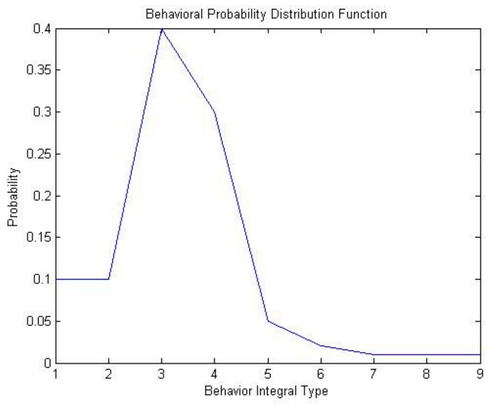
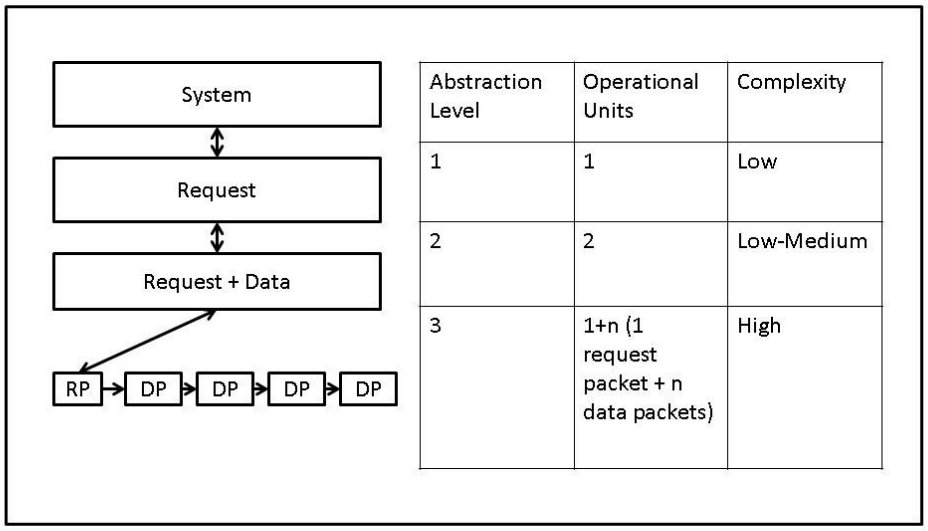


**Figure 2 – Example behavioral PDF**

The classification of an agent as deviant is a two-fold process. The first step involves the individual, or local-scope determination of deviant behavior by each agent. This is computed by calculating the mean probability of a behavior for the entire set of agents. Let $\gamma$ be the number of agents in the system and let $\beta$ represent the number of behaviors present in the system. Let $\eta_{\gamma \times \beta}$ represent a matrix of dimensions $\gamma \times \beta$ containing the historically and temporally-updated probabilities of a certain behavior $\xi$. The local-scope mean probability vector, $\varphi$ is computed for each node $i$ in (1).

$$\varphi_\beta = \sum_{i=1}^{\gamma} \eta_{\gamma \times \beta} \quad (1)$$



| Abstraction Level | Operational Units | Complexity |
|---|---|---|
| 1 | 1 | Low |
| 2 | 2 | Low-Medium |
| 3 | 1+n (1 request packet + n data packets) | High |

Figure 3 – Behavioral Abstraction Level (#1 used)

The vector φ is then analyzed for global and local maxima. Since most of the behaviors will likely be statistically represented by a larger maximum peak, deviation in the system behavior will likely manifest itself eventually over time as a smaller, local maximum, as seen in Figure 4. This smaller local maximum can be correlated to a particular behavior, ξ, as the maxima-finding algorithm is set to return a discrete location of the occurrence of the maxima.

The agent corresponding to the maximally-defined deviant behavior $\xi_d$ is then found by analyzing the column of data in the probability matrix $\eta_{\gamma \times \beta}$ corresponding to $\xi_d$ and then finding the row within that column containing the maximum value for the given $\xi_d$ with in a certain tolerance value τ, representing a probability value.

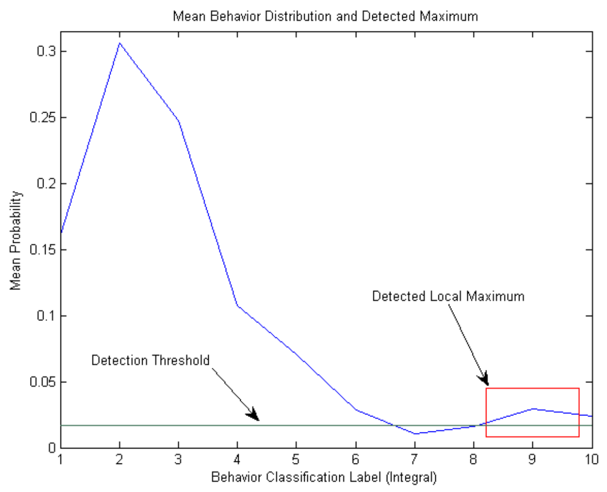


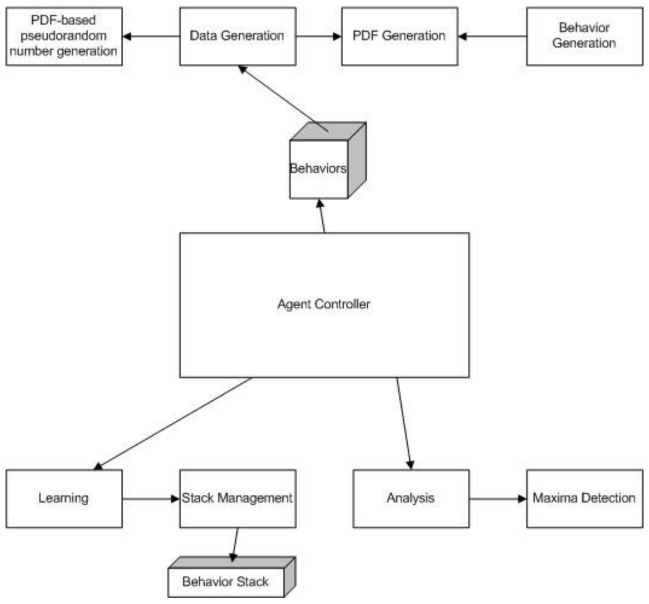
**Figure 4 - Detected Local Maximum - possible indicator of deviant behavior**

The detection strategy implemented by MDS therefore allows for the detection of a single anomaly within a minimal requirement for time – a threshold is used to detect whether the local maximum should be flagged as suspicious or not.

## Ordering of Data

MDS, because of its maxima detection mechanism, has an inherent requirement for normalization of input data. This means that all the theory and methodology presented here assumes Gaussian or other normalization of the data set. Figure 4 shows common data represented by a Chi-Squared distribution, which allows detected maxima to be much more easily distinguished, as far as the eye can see. To an extent, a Chi-Squared distribution will yield optimal results, as "normal" behaviors are skewed to the left (or right) while deviant behaviors, in an ideal circumstance, will be represented on the opposite side of the mean behavioral vector PDF.

Of course, theory differs from actual implementation and operation, and thus deviant behaviors will not always be skewed properly. However, a generalized attempt at ordering is essential to the proper functionality of MDS. The specific ordering methodology is not discussed here, as the result data is generated using a pre-ordered set



(the interactive requests are generated such that the input results will have some form of normalized ordering.) However, it would not be difficult to implement an ordering technique based on simply tracking action/label frequencies and then re-ordering the behavioral frequency column labels appropriately.

**Figure 5 - MATLAB-based implementation system diagram**

## DPC Configuration

Because maxima detection is based on accumulating a statistically significant number of requests over time, having a greater number of nodes in the device network will decrease the time required to stabilize the detected maximum. This is because a greater collection of nodes yields more request data points and subsequently more uniform average representation. Therefore, the smaller the node collective, the more DCCs are required per DPC for stabilization purposes. Furthermore, more DPCs will need to be run to converge accurately. This behavior is analyzed in the hybridized IDS chapter.

## Implementation

The MDS was originally implemented in an exploratory phase using MATLAB 7 to determine its viability and performance. The detection system was comprised of nine core modules, consisting of an agent controller execution program, followed by data handling units, machine learning aspects, the maxima detection system itself, and the simulated data generation portion. These components can be seen in Figure 5. Two objects maintained the status of the system, and are identified by the shaded, three-dimensional boxes. The first of which is referred to as a "behavior" unit, containing the basic abstraction data including requesting node and the actual request made. The behavior units themselves, corresponding over time to the average system state, were managed though a behavior stack, containing a set of pushed behaviors received from the interconnected nodes. The purpose of the stack was to delay an overall update to the state of the system, allowing for a more stable, characteristic update containing more data points, thereby eliminating the impact of a single data point on the stability of the system. Once a preset number of behaviors was collected in the stack, the stack contents were popped and averaged into the overall system profile. As an intermediary between the average system representation and the behavioral stacks, a matrix representing individual nodes and their behavioral labels is used to maintain the total number of interactions present within the system. Useful for debugging purposes, this history matrix eventually became one of the most important foundations in hybridizing MDS, and the cross-correlating IDS, as will be seen in the section referring to the two-system hybridization.

Because of promising results and a general degree of operability within the simulation environment, MDS was ported to the Java 5 runtime standard, allowing for ubiquitous device integration provided a Java Runtime Environment[11] that could be run on the platform in question. The Java implementation was added only within the context of the hybridized IDS, so a free-form Java implementation of the MDS is not available for evaluation as it relies heavily on an implementation framework comprising the two systems, and derived from the cross-correlative IDS to be discussed in the next section.

## Performance

This section will discuss the performance of MDS with respect to deviant agent pervasion – the density of malicious nodes expressed as a percentage of the homogeneous device network. In order to maintain similar metrics through the course of this document, the original MATLAB implementation will not be used as the benchmark case application; rather, the efficacy of MDS will be analyzed as a performance component of the hybridized IDS discussed in Chapter 5.

Because MDS can only detect at most one malicious agent, it does not make sense to discuss MDS performance in the case of more than one anomalous agent per device network. This constraint leads to a very small measurable amount of description and performance evaluation, so the metrics utilized for MDS performance will relate the number of fluctuations MDS undergoes during its detection algorithm as the pervasion density increases. To clarify this a bit, consider a case in which 30 device nodes exist on

[11] JRE

the homogeneous device network. If on this network, there exists a malicious node pervasion of 20%, (i.e., six deviant agents exist), MDS will detect at least one of the six nodes within its first iteration. However, during subsequent cycles, the exact detected node may shift between any of the six potentially deviant nodes. This is expected behavior, since the local maximum will shift among the various deviant nodes as time progresses and more data is gathered.

Because the most viable MDS implementation exists within the hybrid IDS context discussed in Chapter 5, the discussion and data generated for and collected from MDS mechanisms will be sourced from the hybrid IDS scenario. Despite this, the performance and specifications are unique, in component context, to MDS behaviors and system performance. For more information regarding actual data obtained from the MDS component of the hybridized system, the reader is directed to Appendix A for raw data and performance statistics.

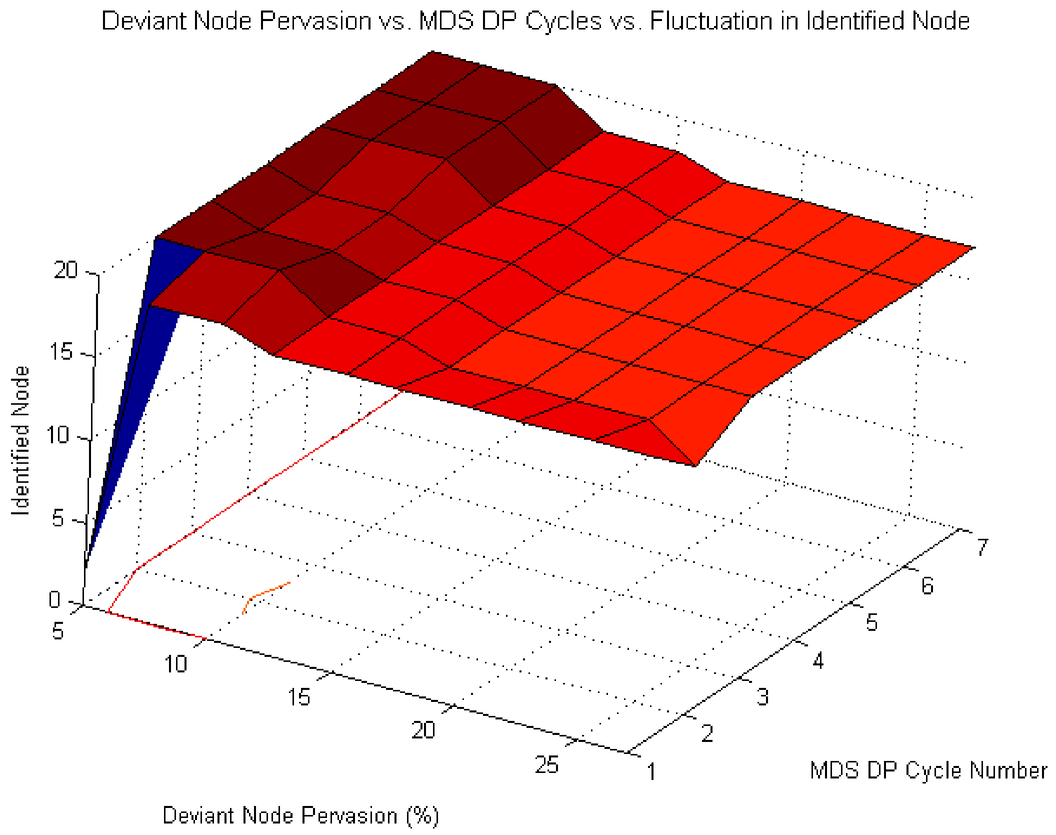


**Figure 6 - Variation of the single detected node number over time (as DPCs increase)**

Figure 6 demonstrates that MDS is relatively stable in selecting a node to identify with a fixed node cluster size of 20 agents, with a varying percentage of deviant nodes from five to 27%. The number of nodes representing the deviant nodes always included node number 20, and included more nodes progressively, adding 19, 18, 17, 16 and eventually 15 as the pervasion percentage increased. Ironically, the greatest instability occurs in one of the simplest test cases, yielding an incorrect initial identification of node number 2. This initial identification anomaly shows that it is important to alter the number of MDS cycles based on the context of the application. All other trials identified correct nodes at all times, typically selecting node number *n* or (*n-1*) for a scenario consisting of *n* nodes. This is primarily due to the statistical distribution of the PDF, and is not a characteristic of MDS.

Let it be noted that Figure 6 does not identify the *number of detected nodes*, but rather identifies a *single detected node number* as the MDS progresses through DPCs. It should be mentioned that the deviant node number index value is predetermined by the dataset, and is *not* influenced by position; node number 20 was always detected in this dataset because input conditions always specified the deviant behavior as occurring under node 20, among others, for instance. MDS would just as easily identify a deviant node placed at any other index (e.g., the deviant nodes being represented by node numbers 4, 5 and 6 in the collective of 20. Node 4, 5 or 6 would then be conclusively detected by the MDS.)

# CHAPTER IV

# CROSS-CORRELATIVE IDS

This section of the document will detail the system and methods of the cross-correlative IDS portion, hereby referred to as a CCIDS[12]. Based on using the properties of vector/matrix cross correlation[22], the CCIDS provides features not present in MDS such as a greater response flexibility, and more importantly, the ability to detect multiple anomalies within a collective of nodes. This is accomplished through the implementation of a mathematical cross-correlation operation that assigns scores to individual node behavior averages with respect to the overall system behavior. Like MDS, the CCIDS utilizes the same level of abstraction to represent behaviors in the system. This allows CCIDS to later be integrated and hybridized with MDS and allowing them to share similar historical information that accurately and homogeneously represents the system state. Also like MDS, the CCIDS utilizes a similar data structure, the agent history table, to record and organize input system behaviors for eventual analysis. To see how CCIDS uses this datastructure, let $\mathbf{\Lambda}$ represent a matrix of dimensions $\mathbf{m}$ x $\mathbf{n}$ containing the binned, recorded request histories for $\mathbf{m}$ nodes and $\mathbf{n}$ classification labels. Let $\boldsymbol{\eta}$ represent the row-summed and averaged vector derived from $\mathbf{\Lambda}$ containing an overall probability distribution representing the overall state of the system according to the classification labels. Lastly, let $\boldsymbol{\gamma_i}$ represent a transposed vector containing the *individual* averaged probability distribution of a behavior for a particular node number $i \in m$. The scores $\sigma_i$ from the resulting cross-correlation are obtained by $\sigma_i = \boldsymbol{\eta} \cdot \boldsymbol{\gamma_i} \; \forall \; \boldsymbol{i} \in \boldsymbol{m}$. (2)

[12] Cross-Correlative Intrusion Detection System

The resulting vector containing the cross-correlation scores for each node is analyzed according to a threshold specified in the IDS runtime environment. Although the vector product operations are not significantly process-intensive for smaller number of agents because the cross-correlation is performed on linear vectors, the nature of the embedded platform requires the minimization of unnecessary computations to save on power and resource requirements. To meet this goal, and to allow for the system to experience change on a global basis without severely affecting the model's integrity and anomaly detection resolution, the cross-correlations are performed based on the number of input requests received, regardless of their origins. This allows for the accumulation of a statistically relevant number of request classifications to be added to each correlation run, and minimizes the impact of a smaller-scale anomaly within a node that may not be malicious but rather the result of an unforeseen consequence of the task being processed at the time.

With this control mechanism in place, the scores are analyzed by comparing each score in (2) to an average composite score generated from all the score entries. Should one or more nodes deviate from the average score according to a specified tolerance value, the node is flagged as suspicious and added to a list containing suspected nodes, maintained separately by each device.

## Thresholding

Central to permitting CCIDS deviant node identification, threshold-based detection sets a point at which an individual node score must deviate from the average/composite score to be flagged as suspicious. The selected threshold must be particular to the application context in which the CCIDS is deployed, and thus must be selected manually (or via a hybridized approach as detailed later in Chapter 5). The results of selected thresholding are seen later on in this chapter, in Figure 9.

The initial threshold values, $\psi_0$, selected for CCIDS (and primarily the hybrid approach in Chapter 5) stem from a 100% deviation in the average node score. For this entire document, the dataset in use created an average score, $\sigma_{avg}$, of approximately 0.2. The determination of a deviant node is made if $\psi \geq |\sigma_{avg} - \sigma_i|$ where $\psi$ represents the selected threshold, is true. Therefore, to create the initial threshold $\psi_0$, the value was originally set to 0.2 to represent the 100% deviation. This implies that only scores exceeding the average by plus or minus 100% would register as suspicious. Of course, such a case would be rare, and thus Figures 8 and 9 demonstrate initial thresholds required for detection to converge via CCIDS, implying that 100% deviation is too extreme a condition for most general purposes.

## Implementation

The CCIDS portion was originally implemented in the Java 1.5 framework, with the intention of execution on a lightweight ARM9 development platform. The implementation structure itself is designed to maximize modularity and implementation flexibility. This also permitted integration of MDS to form the hybridized IDS discussed in the next chapter with minimal modifications due to the object-oriented nature of the IDS implementation.

The IDS itself is composed of seven core class modules and a number of 3rd-party helper objects. The modules are broken down as follows:

1. Manager application – responsible for instantiating the IDS environment and managing its operation
2. NodeManager class – responsible for instantiating objects relating to each node's interaction histories. Spawns associated AgentHistoryTable instances, as well as HistoryObjects.
3. HistoryObject – contains a vector mapping request instances for each node for which the IDS is logging activity. For example, if the IDS is running on aircraft A, HistoryObject instances are created for nodes B, C, and so on.
4. AgentHistoryTable – each IDS maintains one such object containing the overall binned histories for all nodes versus all requests. Represented by $\mathbf{\Lambda}$.
5. IDSEngine – This object is instantiated by the Manager to perform single or multiple-anomaly detection based on data contained in the AgentHistoryTable object instance. This unit is the most critical component and analytical tool of the IDS system.
6. ScoreUnit – A helper class used by IDSEngine
7. IOManager – used for file-based or network-based retrieval of requests made to the

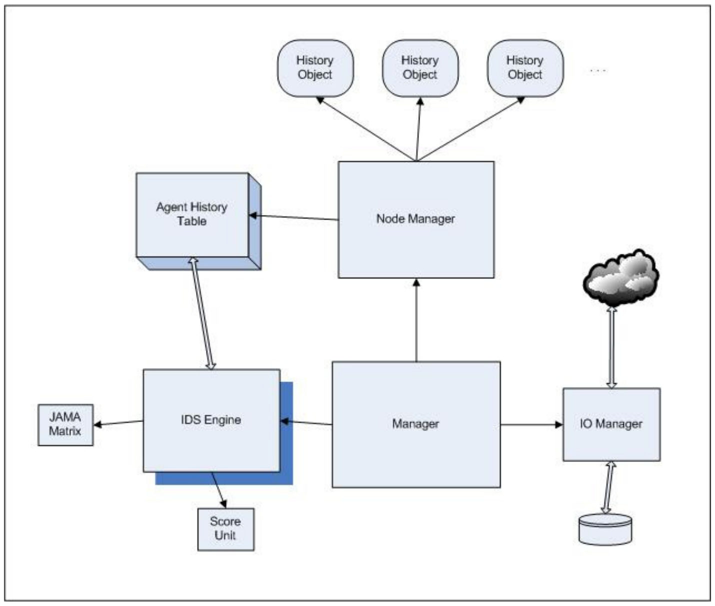


Figure 7 - CCIDS Java component diagram

IDS's host system.

Figure 7 shows the derivative arrangement of the CCIDS, with the Manager application controlling overall execution and instantiating calls to the data collection, data management, and score analysis modules.

## Performance

Unlike MDS, the CCIDS portion does not require a logical ordering of labeled data into a Gaussian or other normalized distribution. This reduces dependence on input data ordering and organization, but results in an extreme dependence on tuning/tolerance factors. While MDS remains sturdier as far as tuning requirements are concerned, CCIDS's efficacy varies greatly based on the selected tolerance values.

To measure IDS performance based on CCIDS performance alone, this section will focus on tuning thresholds required to achieve convergence from the CCIDS. A properly tuned CCIDS mechanism will properly identify the deviant agents within its first performance iteration (DPC). In addition to the tuning thresholds, convergence versus deviant node pervasion (the percentage proportion of deviant

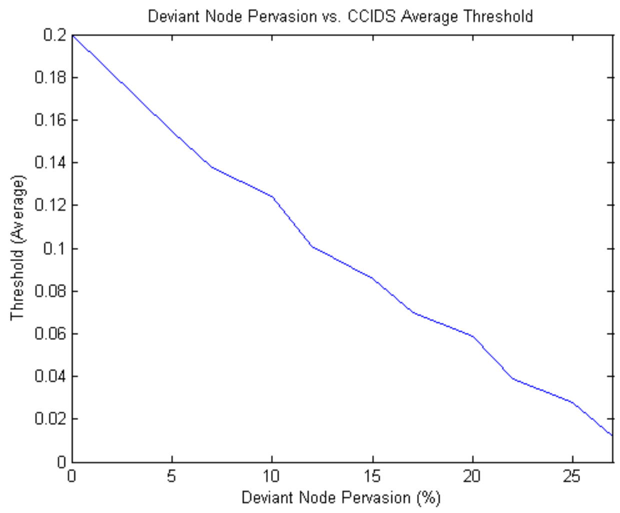


Figure 8 - Average required threshold for CCIDS convergence

nodes vs. all the nodes in the homogeneous device network) becomes a factor in assessing system performance. The reader will note that the data used in these results was generated from the hybrid IDS approach detailed in the next chapter. While the hybrid IDS will alter the tunings for CCIDS dynamically, the data from the upcoming section still represents accurate runtime information, in its component breakdown, for a CCIDS-only implementation provided that only CCIDS data is analyzed in context with the optimized tuning parameters.

Figure 8 represents the average threshold required for convergence of the CCIDS based on a varying pervasion of deviant nodes within the node network. The surface plot, shown in Figure 9, illustrates the varying required threshold for convergence based on not only pervasion, but also the number of nodes in the collective as a whole. The behavior can generally be regarded as linearly dependent on pervasion, not the number of nodes.

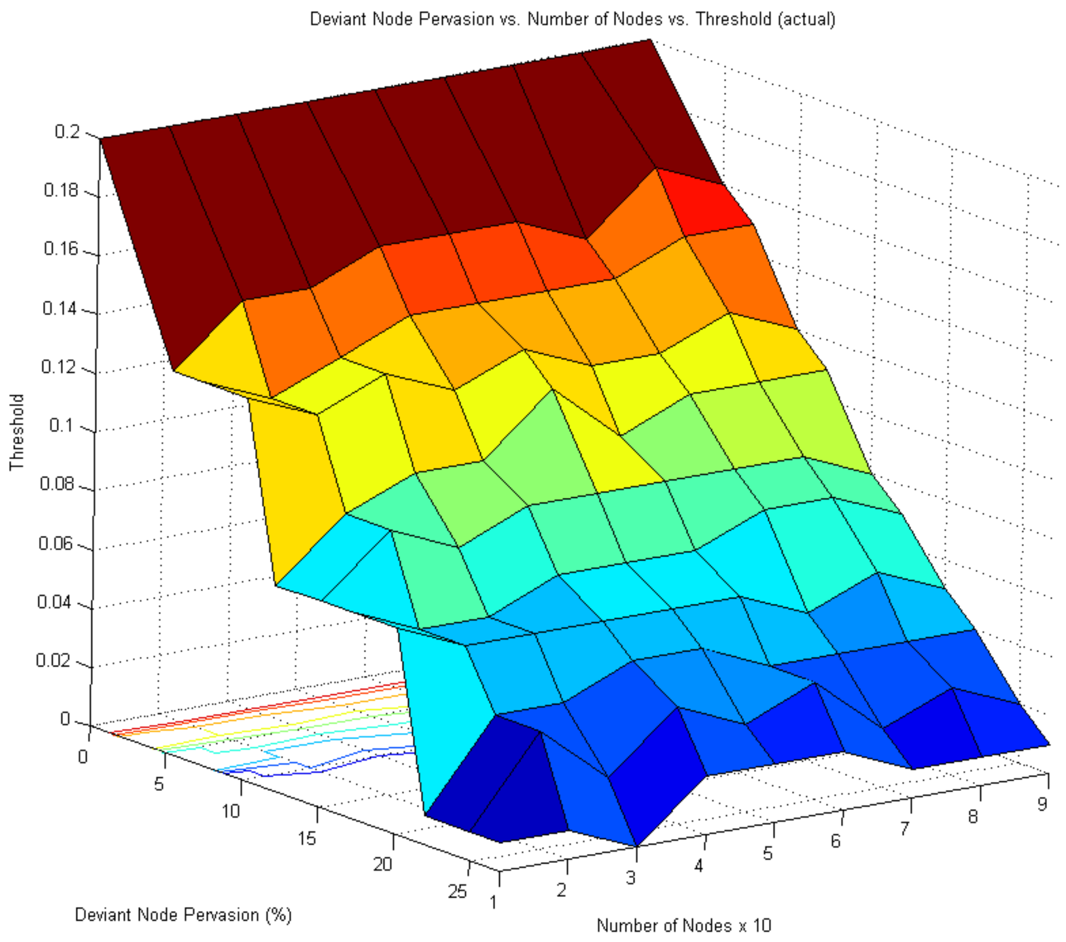


**Figure 9 - Required threshold for CCIDS convergence with varied deviant node pervasion and network size**

It is noted that in some circumstances (when the threshold exceeded 22%, for instance), despite having selected a threshold, CCIDS never converged upon a solution, or did so poorly. For information regarding these cases, the reader is invited to examine trial data presented in Appendix A.

## False Positives

Because of the multiple anomaly detection capability, CCIDS consequently suffers from its tendency to detect false positives more frequently than MDS. Because the criterion for selecting and identifying a malicious node is completely based upon the selection of a proper threshold, this consequently requires careful tuning of this threshold within the execution context of the functioning system. This raises the question of how such a threshold should be applied if false positives are detected within the first execution iteration of the CCIDS. The answer to this lies in the provision of training data and intelligent tuning of the threshold such that only true positives (actual deviant nodes) are found within the behavioral dataset.

Training data is defined quantitatively and proportionally dependent on the number of connected nodes in the collective, and the number of overall behaviors/interactions possible within the system context. For a large number of connected nodes, the behavior, theoretically, becomes established more rapidly, since more devices will be exhibiting similar behaviors. Similarly, a smaller number of behaviors requires less time for the system to stabilize, since the statistical representation of a larger number of behavior classes will take longer to receive data points as the number of behaviors increases towards infinity. This leads to a conclusion that the threshold must be tuned according to several factors present within the system context at initialization time of the homogeneous device network. The complexity arising from the requirement for training data is resolved in the next chapter.

# CHAPTER V

# HYBRID IDS

The individual approaches to homogeneous device network security, as presented by the intrusion detection tactics of MDS and CCIDS, provide a partial solution to the overall issue of identifying deviant nodes in a homogeneous device network. Each system is tailored to provide a particular benefit, such as not needing training data in the MDS case, or providing multiple anomaly detection, in the CCIDS case. However, neither solution can offer the full protection of a combined approach, drawing from the strengths of both systems to surmount their respective weaknesses in a symbiotic manner. This IDS approach will be referred to as a Hybrid IDS or HybrIDS for the purposes of this document.

The primary principle governing the operation of HybrIDS consists of the sequential operation of MDS and CCIDS. More specifically, the lack of temporal requirements for single-anomaly detection specified in MDS can be used to tune the detection threshold for the CCIDS portion of the system. This produces accurate results that are found almost immediately, which can be used to actively remove instances of false positives present in the multiple results from CCIDS. To do so, HybrIDS implements a switching algorithm that determines whether conditions have been met to transition from the primary to secondary stage of the IDS (MDS to CCIDS). This algorithm will be referred to as the Hybrid State. The end product of the Hybrid State is a timing value, Tau, which determines how many DCCs are required before transition from first to second stage.

The Hybrid State is an elementary data structure that computes the value of Tau in DCCs by taking into account three critical system components that are the most influential in determining transition requirements for the homogeneous device network. The first is the number of connected nodes. As the number of nodes $\boldsymbol{\gamma}$ increases, there is a higher likelihood that an increased device presence will stabilize the overall system behavior. The second component involves the number of overall system behaviors present in the system. As the number of behaviors increases, so does the time (in cycles) for all the behaviors to experience a representative number of data points. If $\boldsymbol{\beta}$ represents the number of behaviors present in the system (represented by a set of behavior-separated bins into which collected data points can fall into), and $\boldsymbol{\sigma_0}$ represents the number of DCCs required for an average stabilization, then the function $\sigma_0 = f(\beta) | \beta \to \infty$ is a constantly increasing function. The third component is a variable function, $\boldsymbol{g(\gamma)}$ that returns a constant multiplicand that modifies the effect of the two prior systems to determine Tau in terms of $\boldsymbol{\sigma_0}$. The resulting Tau in number of DCCs $\sigma_0$ can be expressed as:

$$\tau = k_0 \times \beta \times g(\gamma)$$

where $k_0$ is a pre-determined software-related constant issued before run-time and where the

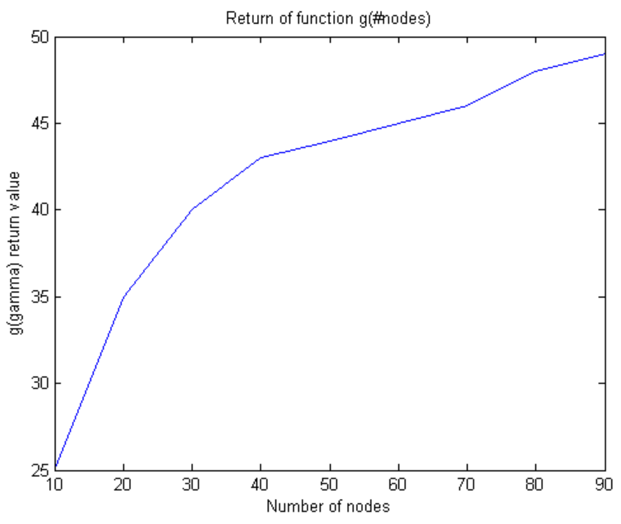


Figure 10 - Represents the returned gamma function value to yield IDS transition

function $\boldsymbol{g(\gamma)}$ can be expressed by a non-linear function based on the number of input node agents. Seen in Figure 10, the function was derived experimentally and approximates a logarithmic increase, such that agent groups with larger numbers of nodes do not immediately transition IDS stages.

The returned value of $\boldsymbol{g(\gamma)}$ allows the overall function $\tau(k_0, \beta, \gamma)$ to exhibit a surface of values for the Hybrid State in terms of $\boldsymbol{\sigma_0}$ as shown

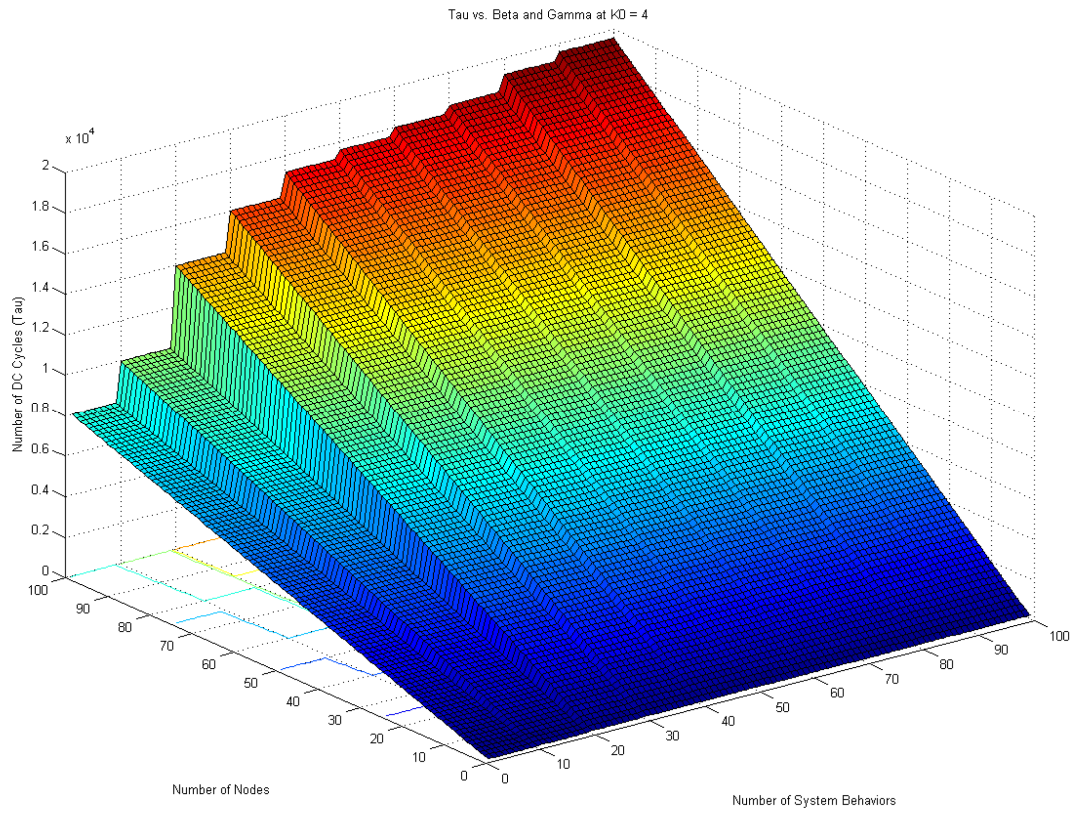


**Figure 11 - Surface of possible Tau values (for IDS transition) versus number of nodes (Gamma) and number of behaviors (Beta)**

in Figure 11 when the value of $k_0 = 4$ (selected to reduce runtime and increase accuracy.) Because of an algorithm implemented in CCIDS, the number of DPC executions performed is proportional to the system behaviors and nodes as well, so while there is a significant overall increase in the number of DC cycles as nodes increase, the number of DCCs per DPC decreases. This can be seen in Figure 12.

## Hybrid State Control Flow and MDS/CCIDS Transition

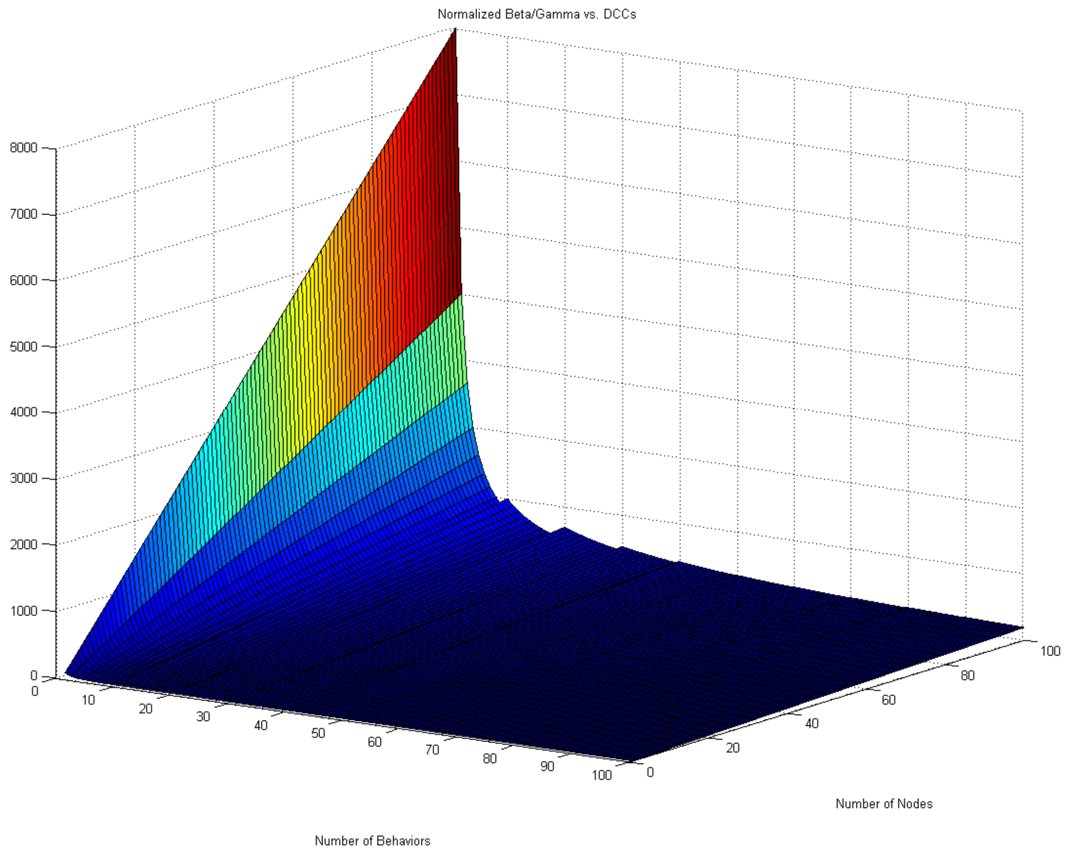


**Figure 12 - Normalized surface representing IDS transitions based on selected Tau (dependent on environmental configuration)**

HybrIDS relies on the Boolean state of the Hybrid State object. Should the correct number of DCCs have passed according to the Tau function detailed in the previous section, the MDS state will be *false* and the CCIDS state will resolve to *true*. Following this state change, the IDS will begin a transitory phase in which the long-term results of MDS are used to calibrate the evaluative results of the initial set of CCIDS iterations.

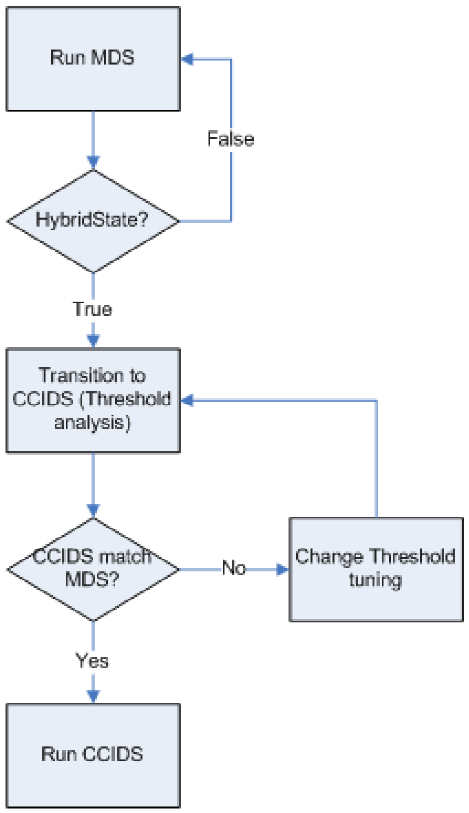


Figure 13 - IDS transition logic flowchart

This transitory phase involves an output suspected agent vector, λ, which contains the findings of the MDS phase. It is noted that $\max(\dim(\lambda)) = 1$, since MDS can at most yield one suspected agent vector. Else, λ may be null. Also critical to the transitory phase is the suspected agent vector ξ, which contains the evaluative findings of the CCIDS phase. Given the set of all ***n*** possible agent nodes, $\boldsymbol{\alpha_n}$, the relationship between ξ and **α** is such that $\xi \subseteq \alpha_{n-1}$. This implies that the maximal set of possible deviant nodes can be some or all of the connected nodes except for one, which must exist for the cross correlation to have any meaning. Given sets λ and ξ, the transition phase involves the constant changing of the tuning threshold until the condition $\lambda \subseteq \xi$ is satisfied. The threshold value begins at a default state and is tuned either positively or negatively until the desired subset condition is reached. The logical flow of the transitioning mechanism can be seen in Figure 13.

## Implementation and Architecture

HybrIDS is implemented according to the Java 2 version 1.5 Standard Edition API, according to the design and previous implementation of CCIDS. In fact, the hybridization component and MDS engine were added onto the existing CCIDS framework, though their integration effects a critical and fundamental change in the nature and properties of the system. Many concepts and execution primitives from CCIDS were maintained, and added to the Java port of the MDS portion of the HybrIDS. The resulting framework yielded a number of important properties: modularity, homogeneity, and a shared data infrastructure.

The most significant changes to the architecture is the addition of the MDS engine, and the conversion of the primary IDS Engine to the CCIDS subcomponent, as shown in Figure 14. Other minor changes include changes to the IDS management system and the application management system, both of which were altered to allow for IDS phase transitioning and sequential execution. Sequential execution is still governed by the same DCC/DPC cycle management scheme originally developed for the stand-alone CCIDS.

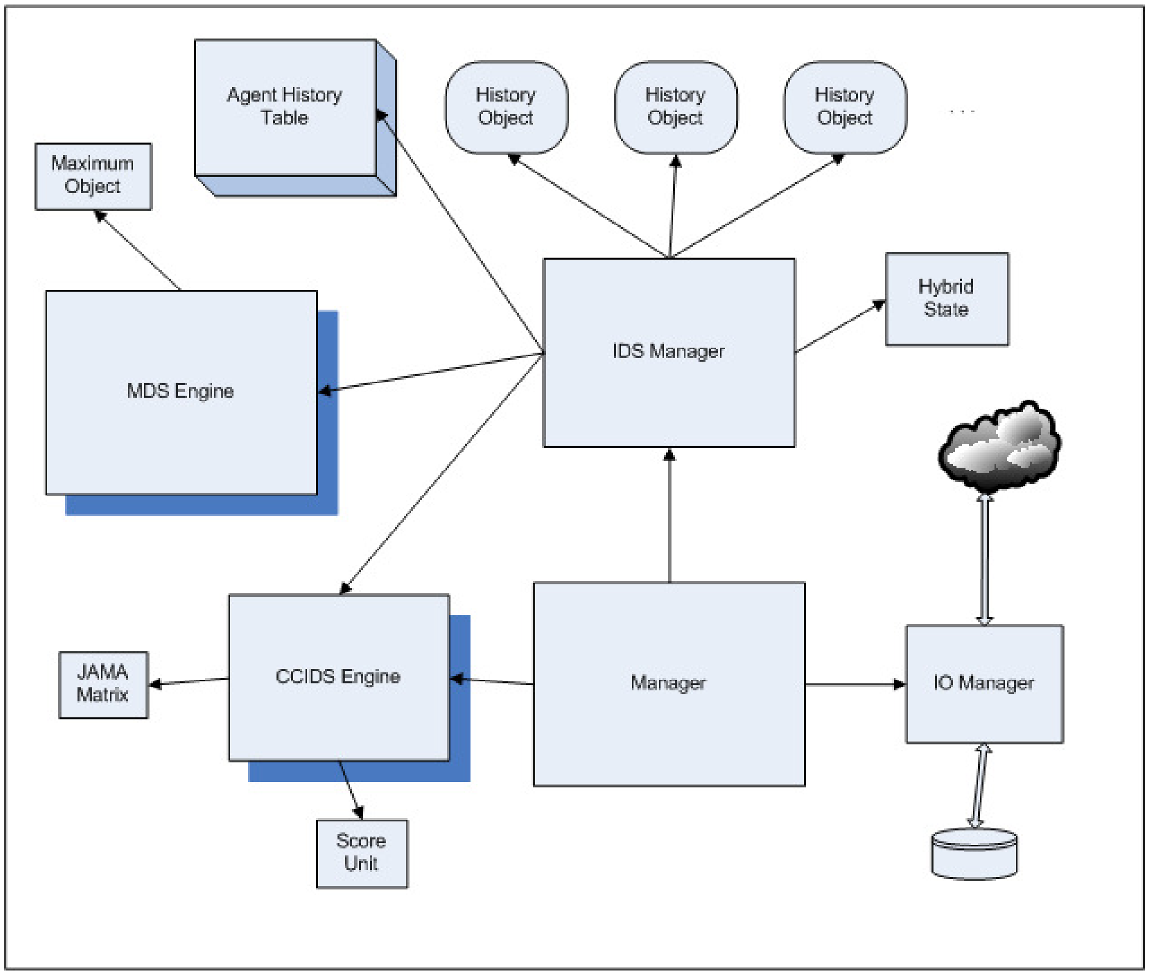


Figure 14 - HybrIDS Java component diagram

One of the largest benefits of the Java-based implementation and concurrency with previous development practices is the resulting data sharing occurring between the two IDS systems – the sharing of the Agent History Table. This data structure, originating from the CCIDS Java implementation, but also present in the original MATLAB 7 implementation of the MDS, contains the machine-learning elements critical to temporal and statistical system behavior determination. A single instance of this object is created and passed by reference to the various subcomponents and engines of the IDS, minimizing the required memory footprint. Let θ represent the resident memory size of the Agent History Table in bytes, and ε represent the memory footprint of a 64-bit double datatype. The total resident memory size is then

$$\theta = \beta \times \alpha_n \times \frac{\varepsilon}{8}$$

For a typical agent history table consisting of 35 agents and 10 behaviors, the memory footprint of the associated IDS's Agent History Table would be a maximum of 2.73 kilobytes. Because the Agent History Table is the most memory-intensive portion of the entire IDS, maintaining the historical and machine learning components required to track system behaviors, it is easy to see why this HybrIDS model is extremely adaptable to real-time and embedded system architectures, where memory and computational resources are at a minimum. Further design considerations include compact compiled application size (compiled as a Java JAR file), not exceeding 46 kilobytes of required storage space.

## Performance

The as-tested performance of the HybrIDS showed significant improvements in the detection accuracy over the single IDS case of either MDS or CCIDS. The improvements were so vast that each and every system trial resulted in a 100% accurate detection at the transition intervals selected within a certain range of deviant node pervasion. The number of transitioning iterations and number of iterations before accurate detection for either MDS or CCIDS were utilized as performance metrics to evaluate the efficiency of the HybrIDS. The test scenarios varied in the percentage of malicious node pervasion, as well as the number of nodes used in the test. An overall figure representing the total number of DPCs for all portions (MDS, CCIDS and transition) was also included during evaluation. Approximately 383MB of trial scenario data was generated to be used as the basis for inter-node device requests seen from the perspective of a single node.

Two sets of graphs will be presented in this section: The first set contains three

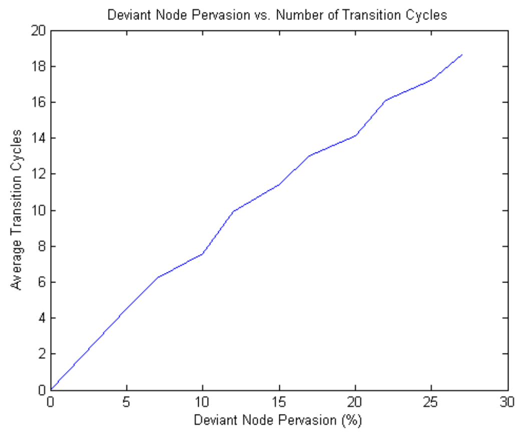


Figure 15 - Average transition cycles vs. percentage of deviant nodes

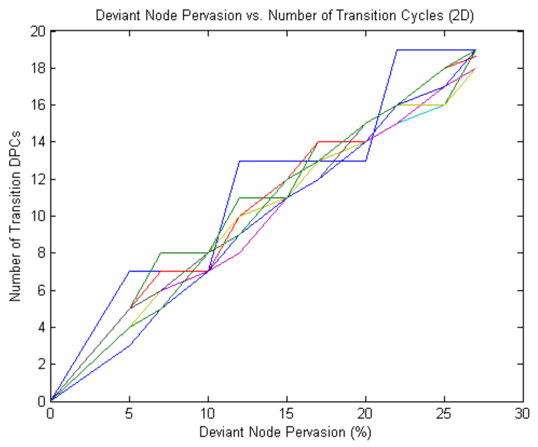


Figure 16 - Enumerated transition cycles vs. percentage of deviant nodes

graphs with data about the number of tuning cycles required between the MDS and CCIDS phases such that CCIDS may be properly tuned to avoid false positives while accurately detecting the multiple anomalous nodes. The graphs will display this information as 1.) an average, 2.) as an interpolated multi-trace plot, and 3.) as an interpolated three-dimensional surface plot. The second set of graphs will represent in various ways the total number of DPCs consumed by the entire IDS process, including the MDS, CCIDS and transition portions. The same graph methodology from the first set will also be observed.

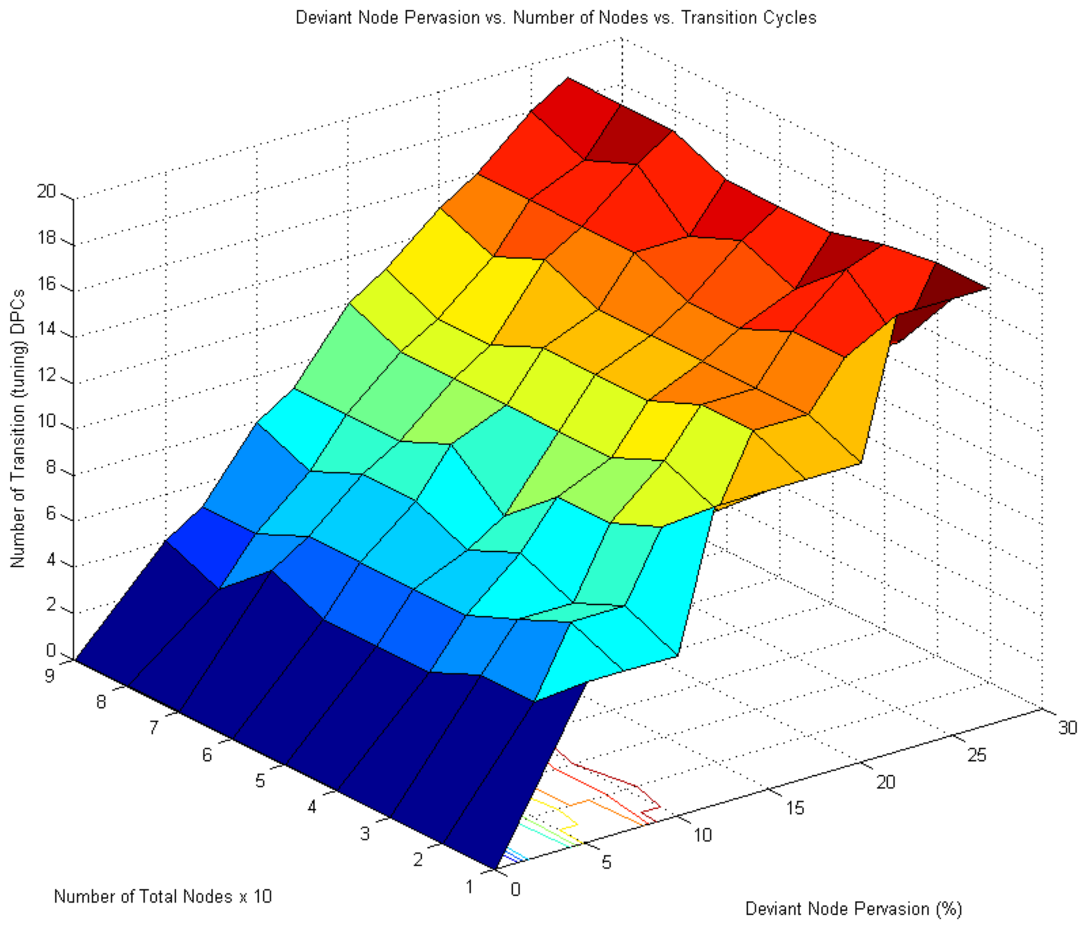


Figure 17 - Surface of required transition cycles vs. percentage of deviant nodes and size of network

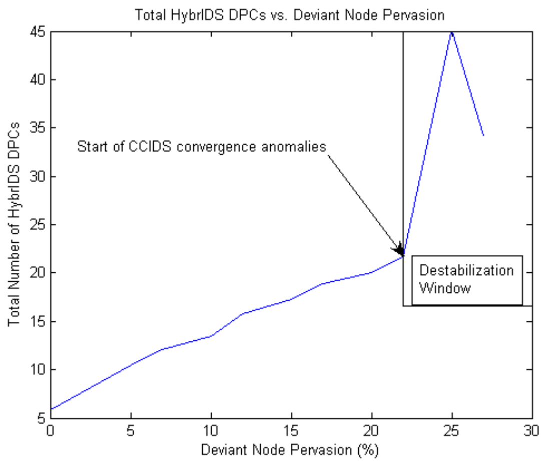


Figure 18 - Average number of DPCs (total) vs. deviant node pervasion

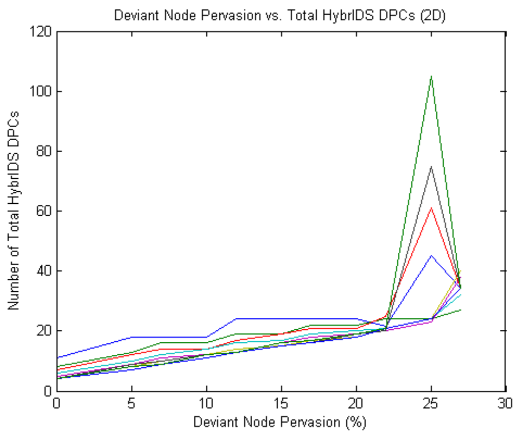


Figure 19 - Enumerated number of DPCs (total) vs. deviant node pervasion

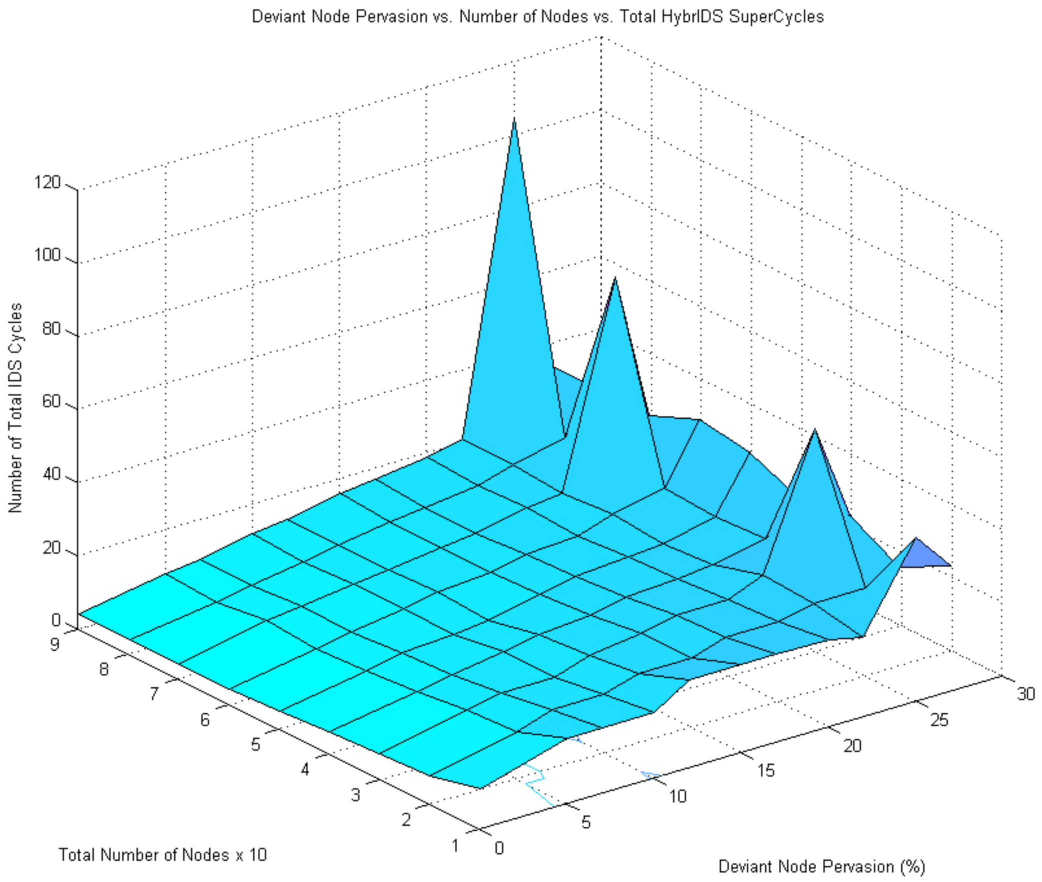


Figure 20 - Surface of total IDS DPCs vs. percentage of deviant nodes vs. total network size; High peaks occurring above 22% pervasion indicate significant instability/low rate of convergence. Some non-converged values were averaged.

Figure 17 demonstrates that the number of tuning cycles necessitated by a particular concentration of deviant nodes within the device network is based almost exclusively and linearly by the percentage of deviant nodes. The number of transition cycles is also linearly dependent on the starting value for the CCIDS threshold, which is not seen in any of the figures. It is feasible to reduce the number of overall iterations by starting with a lower threshold, but this may be ill advised given that some CCIDS threshold are very close to the starting threshold value of 0.20. Figure 15 demonstrates, when taken into account with Figure 16, that the number of DPCs dedicated to tuning CCIDS is relatively independent of the total number of agents on the device network. In this case, the single-most deciding factor is the percentage of pervasion. It is notable that there are some differences between the behavioral curves when considering the number of tuning cycles, which can be attributed to non-convergence in a few particular cases, necessitating an interpolation so that the graphs may be comparable in nature.

During the data generation phase, trials were generated with differing numbers of deviant agents, to create the different pervasion scenarios depicted in the figures. Because the number of agents must correspond to an integral number, it is noted that not all of the percentages are exact. To illustrate this, given a collective of 30 nodes with a 15% pervasion, the returned requirement of the test data indicates a need for 4.5 deviant nodes. Since a fractional node is an impossible scenario, in this case, the actual selected dataset corresponded to 5 deviant nodes out of a total of 30 nodes. This is relatively close to 16.7%, which means that the computation of 30 nodes for the 17% pervasion case also computed 5/30 deviant nodes. This explains some of the discrepancies in Figure 18, especially where the stair-step-like behavior of some of the cluster sizes is observed. The stair-step nature is due mostly to some replication of scenario data between pervasion percentages. A uniform rounding and computation algorithm was applied to indicate which trial datasets should be used, such that the overall test scenario would tend to behave more or less the same way, despite replications among datasets.

Figure 20 shows a three-dimensional surface representation, interpolated, of the number of transition cycles required to set MDS properly. This surface yields a more palpable view of the pervasion density impact on the number transition cycles required to tune the HybrIDS.

Figures 18 and 19 paint a very different picture from the transition cycle graphs. Here we see the real-world effects of the higher pervasion percentages. Because the tuning-based graphs will always display a somewhat linear response, due to a threshold limit of how low or high the IDS can be tuned, non-convergent systems will still display an upper bound in terms of tuning cycles. This is not the case with the overall system behavior. As seen in Figure 19, where it is labeled "Start of CCIDS convergence anomalies", the number of total cycles begins to deteriorate with respect to previous system performance. This is in a large part due to CCIDS tuning factors not being able to go lower – in essence, the CCIDS is "on its own" despite MDS's best efforts to calibrate it. In some cases, despite the most flexible tuning, CCIDS can converge, but only after an unusually long and generally impractical period of time. The symptoms of this begin right around 22% pervasion, and fluctuate significantly as the system approaches 27% pervasion. In other cases, the CCIDS portion simply did not converge at all. Because of plotting software, discontinuities are unsupported, and therefore average total cycle response times were interpolated into cases which did not converge. Please see Appendix A for a complete table detailing cases in which convergence was not possible.

In several cases, the convergence period became unreliably large – sometimes exceeding 100 DPCs (a number indicating that ten or a hundred thousand data points have been accumulated before convergence has resulted.) Because data processing cycles are dependent on the influx of a relatively large number of input data, a detection convergence requirement of this many DPCs is generally deemed unacceptable. This arises from the fact that it is difficult to tell whether or not a dataset would converge or not after so many cycles, representing a generally long period of time during a homogeneous device network's "mission span."

## Hybridized Outlook and Discussion

As described earlier in the CCIDS chapter of this document, one of the goals in terms of system-wide efficiency is the independence of the system from the number of devices present in the system. This goal is met by MDS, and further continued by the HybrIDS version, as seen in Figure 20. It is apparent that as the number of nodes increases, there is generally no increase, but rather even a slight decrease, in the number of DPCs required for convergence. Other expected findings, such as the increase in transition cycles corresponding to changes in pervasion, are to be expected. When viewing this from a broader perspective, the tuning threshold has a maximum possible impact on the system's overall convergence time requirement. Therefore, in the long run, the transition performance is not as detrimental with respect to a worst-case scenario as the individual performance of CCIDS.

For the scenario setup described in the concepts and CCIDS sections, it can be concluded that the HybrIDS mechanism is effective, therefore, with pervasion percentages reaching approximately 22% before system-wide behavior and response becomes non-deterministic, or at least non-representative of a reasonable performance envelope. There is some room for improvement, as tuning specific methods and mechanisms in both MDS and CCIDS phases can yield a higher degree of sensitivity, situational forgiveness, and accuracy. One such proposed method would be to add a method of tracking the positives that are detected. It is very worthwhile to note that during the CCIDS portion, all of the malicious nodes were properly identified at all times. It is the inclusion of false positives for excessive periods of time that defined whether, for testing purposes, the system converged or not. Therefore, by keeping track of the dynamic nature of the positives identifications generated, false-positives included, it is theoretically possible to eliminate contenders representing the false positive occurrence from the suspected node list based on its temporal manifestations on said list.

Given its observed behavior using the generated datasets, HybrIDS demonstrates its adaptability to the embedded device platform. By representing interactions with a scheme employing a high level of abstraction, it minimizes computational intensity through mediation via DPC data management and transitional IDS process mediation. It is capable of accurately identifying anomalous networked nodes with a pervasion density of up to 22%, and is scalable to a large number of networked nodes with minimal impact on response time and performance. Finally, a compact implementation form factor (46 kilobytes when compiled as a JAR file) coupled with a small memory footprint (the largest data structure occupies 2.73 kilobytes) and a ubiquitous port and runtime environment as supplied by the Java 1.5 standard, ensures seamless integration and a large degree of applicability to various device classes and categories.

# CHAPTER VI

# CONCLUSION

The increasing presence of specialized, embedded devices within the context of a networked scenario, such as in the case of a collective of specialized autonomous vehicles, including tactical and civilian aircraft, automobiles and aquatic vehicles, requires an updated perspective on security and network integrity protection. To this end, equipping traditional methods of data confidentiality, source authentication and data integrity with methods of intrusion detection can bolster the security of networked agents, especially in the case scenario of a network of homogeneous device nodes.

Important to integrating an IDS to an embedded device architecture is creating a system methodology in which a high level of operational abstraction provides a contextual detachment and an isometric system of analysis between all nodes of the device network. Likewise, a small system footprint (both in memory and executable storage space) along with a reduction in the number of overall computations is required to satisfy power requirements and computational resource limitation. The single and hybrid IDS systems outlined in this thesis represent a combinational approach to meeting the requirements stated above. By utilizing a small memory footprint, and discretizing processor-intensive tasks to deterministic time points, the IDSs provide an optimized approach to lightweight intrusion detection.

MDS and CCIDS represent cases where two different approaches have respective strengths in resource utilization, a requirement or lack thereof of training data, and the ability or lack thereof of multiple agent detection. MDS has demonstrated its speed in detecting the presence of an intrusion, without the need for training data, and in a manner that almost always detects the intrusion accurately. When tuned properly, MDS proves to be the most effective in finding the occurrence of an intrusion, when presented with a homogeneous device network and a pre-existing set of behaviors and network size. MDS takes its inherent speed from a thresholded analysis of an averaged PDF representing the overall state of the system. Because the detected local maximum, excluding the global maximum, are only representative of one node, MDS cannot be used to detect the presence of more than one deviant node in the network. This does not exclude the fact that vacillations in the detected suspect node can be used to appropriately identify potential offenders. However, because of the nature of the PDF distribution, a reasonable number of MDS cycles is required to yield the desired suspected agent resolution. HybrIDS recognizes this requirement and bases the number of MDS iterations on the speed of accumulated data as the system receives request inputs.

In contrast to MDS, CCIDS resolves the single-detection deficiency by providing multiple suspected agent resolution. This comes at the cost of requiring precise tuning of internal thresholding levels to achieve accurate detection. CCIDS is flexible to responses in system changes as a mission might update. However, because of the tuning requirement, CCIDS must be provided with an array of sample training data that may not be available in a dynamic system-wide execution context. The lack of this training data yields an unstable and non-deterministic IDS strategy that on its own is incapable of providing significant results to IDS functionality.

With these various strengths and weaknesses, it is therefore logical to consider a hybridized approach, in an attempt to reduce the deficiencies of either system by using combined identification capabilities to provide an overall solution to intrusion detection. HybrIDS provides this solution by integrating the single intrusion detection, high-speed case as a reference point to tune the secondary IDS stage, making use of the multiple anomaly detection capabilities that work well when tuned properly by the first stage. A transitioning system allows for the system to perform in a deterministic, expected manner. As the data demonstrates, an increase in the number of connected nodes does not contribute to an increase in overall execution and convergence time. This yields an important advantage for scalability reasons. The only performance penalty comes in the form of advanced pervasion, affecting only the number of DPCs required for tuning the CCIDS stage. The overall convergence and runtime is largely unaffected.

Coupled with a small executable file size, in a platform-independent implementation utilizing minimal memory resources for optimal resource management, the HybrIDS approach yields a practical IDS methodology applicable to a range of embedded devices within the networked context. Together with existing intrusion prevention mechanisms such as encryption, authentication and signature methods, the proposed HybrIDS can provide an extra, necessary level of dynamic protection to both established and yet undeveloped embedded device network architectures.

# APPENDIX A

## TRIAL RUN DATA

| | 0% Deviant Agents Category | | | |
|---|---|---|---|---|
| #Nodes | MDS DPCs | CCIDS DPCS | HybrIDS DPCs | Transition Cycles |
| 10 | 10 | 1 | 11 | 0 |
| 20 | 7 | 1 | 8 | 0 |
| 30 | 6 | 1 | 7 | 0 |
| 40 | 5 | 1 | 6 | 0 |
| 50 | 4 | 1 | 5 | 0 |
| 60 | 3 | 1 | 4 | 0 |
| 70 | 3 | 1 | 4 | 0 |
| 80 | 3 | 1 | 4 | 0 |
| 90 | 3 | 1 | 4 | 0 |

| 5% Deviant | | | | 7% Deviant | | | |
|---|---|---|---|---|---|---|---|
| MDS | CCIDS | HybrIDS | T-Cycles | MDS | CCIDS | HybrIDS | T-Cycles |
| 10 | 1 | 18 | 7 | 10 | 1 | 18 | 7 |
| 7 | 1 | 13 | 5 | 7 | 1 | 16 | 8 |
| 6 | 1 | 12 | 5 | 6 | 1 | 14 | 7 |
| 5 | 1 | 10 | 4 | 5 | 1 | 12 | 6 |
| 4 | 1 | 9 | 4 | 4 | 1 | 11 | 6 |
| 3 | 1 | 8 | 4 | 3 | 1 | 10 | 6 |
| 3 | 1 | 9 | 5 | 3 | 1 | 10 | 6 |
| 3 | 1 | 7 | 3 | 3 | 1 | 9 | 5 |
| 3 | 1 | 8 | 4 | 3 | 1 | 9 | 5 |

| 12% Deviant | | | | 15% Deviant | | | |
|---|---|---|---|---|---|---|---|
| MDS | CCIDS | HybrIDS | T-Cycles | MDS | CCIDS | HybrIDS | T-Cycles |
| 10 | 1 | 24 | 13 | 10 | 1 | 24 | 13 |
| 7 | 1 | 19 | 11 | 7 | 1 | 19 | 11 |
| 6 | 1 | 17 | 10 | 6 | 1 | 19 | 12 |
| 5 | 1 | 16 | 10 | 5 | 1 | 17 | 11 |
| 4 | 1 | 13 | 8 | 4 | 1 | 16 | 11 |
| 3 | 1 | 14 | 10 | 3 | 1 | 15 | 11 |
| 3 | 1 | 13 | 9 | 3 | 1 | 15 | 11 |
| 3 | 1 | 13 | 9 | 3 | 1 | 15 | 11 |
| 3 | 1 | 13 | 9 | 3 | 1 | 16 | 12 |

| 17% Deviant | | | | 20% Deviant | | | |
|---|---|---|---|---|---|---|---|
| MDS | CCIDS | HybrIDS | T-Cycles | MDS | CCIDS | HybrIDS | T-Cycles |
| 10 | 1 | 24 | 13 | 10 | 1 | 24 | 13 |
| 7 | 1 | 22 | 14 | 7 | 1 | 22 | 14 |
| 6 | 1 | 21 | 14 | 6 | 1 | 21 | 14 |
| 5 | 1 | 19 | 13 | 5 | 1 | 20 | 14 |
| 4 | 1 | 18 | 13 | 4 | 1 | 19 | 14 |
| 3 | 1 | 17 | 13 | 3 | 1 | 18 | 14 |
| 3 | 1 | 16 | 12 | 3 | 1 | 19 | 15 |
| 3 | 1 | 16 | 12 | 3 | 1 | 18 | 14 |
| 3 | 1 | 17 | 13 | 3 | 1 | 19 | 15 |

| 22% Deviant | | | | 25% Deviant | | | |
|---|---|---|---|---|---|---|---|
| MDS | CCIDS | HybrIDS | T-Cycles | MDS | CCIDS | HybrIDS | T-Cycles |
| 10 | N/C | | 19 | 10 | N/C | | 19 |
| 7 | 1 | 24 | 16 | 7 | 1 | 24 | 16 |
| 6 | 3 | 25 | 16 | 6 | 37 | 61 | 18 |
| 5 | 1 | 21 | 15 | 5 | 3 | 24 | 16 |
| 4 | 1 | 20 | 15 | 4 | 2 | 23 | 17 |
| 3 | 2 | 21 | 16 | 3 | 5 | 24 | 16 |
| 3 | 1 | 20 | 16 | 3 | 54 | 75 | 18 |
| 3 | 2 | 21 | 16 | 3 | 4 | 24 | 17 |
| 3 | 2 | 21 | 16 | 3 | 84 | 105 | 18 |

| 27% Deviant | | | |
|---|---|---|---|
| MDS | CCIDS | HybrIDS | T-Cycles |
| 10 | N/C | | 19 |
| 7 | 1 | 27 | 19 |
| 6 | N/C | | N/A |
| 5 | 9 | 32 | 18 |
| 4 | 16 | 38 | 18 |
| 3 | 19 | 40 | 18 |
| 3 | N/C | | 19 |
| 3 | N/C | | 19 |
| 3 | N/C | | 19 |

## REFERENCES

1. Zhenwei, Y., J.J.P. Tsai, and T. Weigert, *An Automatically Tuning Intrusion Detection System.* Systems, Man and Cybernetics, Part B, IEEE Transactions on, 2007. **37**(2): p. 373-384.

2. Freeman, W. and E. Miller. *An experimental analysis of cryptographic overhead in performance-critical systems*. in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999. Proceedings. 7th International Symposium on*. 1999.

3. He, G. and S.R. Tate. *Efficient Authenticated Key-Exchange for Devices with a Trusted Manager*. in *Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on*. 2006.

4. Yang, Q., et al. *An embedded RSA processor for encryption and decryption*. in *ASIC, 2001. Proceedings. 4th International Conference on*. 2001.

5. Seung-Jo, H., O. Heang-Soo, and P. Jongan. *The improved data encryption standard (DES) algorithm*. in *Spread Spectrum Techniques and Applications Proceedings, 1996., IEEE 4th International Symposium on*. 1996.

6. Chih-Chung, L. and T. Shau-Yin. *Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter*. in *Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on*. 2002.

7. Naess, E., et al. *Configurable middleware-level intrusion detection for embedded systems*. in *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*. 2005.
8. Borsc, M. and H. Shinde. *Wireless security & privacy*. in *Personal Wireless Communications, 2005. ICPWC 2005. 2005 IEEE International Conference on*. 2005.
9. Guyot, V. *Using WEP in ad-hoc networks*. in *Wireless and Optical Communications Networks, 2006 IFIP International Conference on*. 2006.
10. Bittau, A., M. Handley, and J. Lackey. *The final nail in WEP's coffin*. in *Security and Privacy, 2006 IEEE Symposium on*. 2006.
11. Cabrera, J.B.D., C. Gutierrez, and R.K. Mehra. *Infrastructures and algorithms for distributed anomaly-based intrusion detection in mobile ad-hoc networks*. in *Military Communications Conference, 2005. MILCOM 2005. IEEE*. 2005.
12. Brutch, P. and C. Ko. *Challenges in intrusion detection for wireless ad-hoc networks*. in *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*. 2003.
13. Dwen-Ren, T., T. Wen-Pin, and C. Chi-Fang. *A hybrid intelligent intrusion detection system to recognize novel attacks*. in *Security Technology, 2003.*

*Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on*. 2003.

14. Kachirski, O. and R. Guha. *Effective intrusion detection using multiple sensors in wireless ad hoc networks*. in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. 2003.
15. Keum-Chang, L. and L. Mikhailov. *Intelligent intrusion detection system*. in *Intelligent Systems, 2004. Proceedings. 2004 2nd International IEEE Conference*. 2004.
16. Mishra, A., K. Nadkarni, and A. Patcha, *Intrusion detection in wireless ad hoc networks.* Wireless Communications, IEEE [see also IEEE Personal Communications], 2004. **11**(1): p. 48-60.
17. Ran, Z., et al. *Multi-agent based intrusion detection architecture*. in *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference on*. 2001.
18. Siraj, A., S.M. Bridges, and R.B. Vaughn. *Fuzzy cognitive maps for decision support in an intelligent intrusion detection system*. in *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*. 2001.
19. Watkins, D. and C. Scott. *Methodology for evaluating the effectiveness of intrusion detection in tactical mobile ad-hoc networks*. in *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*. 2004.
20. Xia, W. *Intrusion Detection Techniques in Wireless Ad Hoc Networks*. in *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International*. 2006.
21. Xiaodong, Z., H. Zhiqiu, and Z. Hang. *Design of a Multi-agent Based Intelligent Intrusion Detection System*. in *Pervasive Computing and Applications, 2006 1st International Symposium on*. 2006.
22. Xiaoqiang, Z., Z. Zhongliang, and F. Pingzhi. *Intrusion detection based on cross-correlation of system call sequences*. in *Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on*. 2005.
23. Yamada, A., et al. *Intrusion detection system to detect variant attacks using learning algorithms with automatic generation of training data*. in *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*. 2005.
24. United States. General Accounting Office. National Security and International Affairs Division., *Air Force rationale for JDAM production decision*. 1997, The Office ;
The Office, [distributor,: Washington, D.C.
Gaithersburg, MD (P.O. Box 6015, Gaithersburg 20884-6015).
25. Daskalakis, C. and P. Martone. *Alternative surveillance technology for the Gulf of Mexico*. in *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*. 2004.
26. Harman, W.H. *ADS-B airborne measurements in Frankfurt*. in *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*. 2002.
27. Hicok, D.S. and D. Lee. *Application of ADS-B for airport surface surveillance*. in *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE*. 1998.

28. Nichols, R., et al. *Testing of traffic information service broadcast (TIS-B) and ADS-B at Memphis International Airport*. in *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*. 2002.

29. Buchegger, S. and J.Y. Le Boudee, *Self-policing mobile ad hoc networks by reputation systems.* Communications Magazine, IEEE, 2005. **43**(7): p. 101-107.

30. Lauf, A. P., Peters, R. A. and Robinson, W. H. *Intelligent Intrusion Detection: A Behavior-Based Approach.* in *Advanced Information Networking and Applications: Symposium for Embedded Computing, 2007. Proceedings. The 21st. 2007.*