AFRL-IF-RS-TR-2003-83
Final Technical Report
April 2003

IMPERISHABLE NETWORKS: COMPLEXITY THEORY AND COMMUNICATION NETWORKING - BRIDGING THE GAP BETWEEN ALGORITHMIC INFORMATION THEORY AND COMMUNICATION NETWORKING

General Electric Global Research

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. AOM100

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-83 has been reviewed and is approved for publication.

APPROVED: 

SCOTT S. SHYNE
Project Engineer

FOR THE DIRECTOR: 

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2003	Final Jun 01 – Dec 02

4. TITLE AND SUBTITLE
IMPERISHABLE NETWORKS: COMPLEXITY THEORY AND COMMUNICATION NETWORKING - BRIDGING THE GAP BETWEEN ALGORITHMIC INFORMATION THEORY AND COMMUNICATION NETWORKING

5. FUNDING NUMBERS
C - F30602-01-C-0182
PE - 62301E
PR - FTNP
TA - M1
WU - 00

6. AUTHOR(S)
Stephen F. Bush, Scott Evans, and Amit B. Kilkarnni

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
General Electric Global Research
One Research Circle
Niksayuna New York 12309

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-83

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Scott S. Shyne/IFGA/(315) 330-4819/ Scott.Shyne@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The most significant result from this project has been experimental validation that complexity plays a critical role in information assurance and can be broadly applied as the basis for security analysis and fault tolerant network design. Complexity Theory is a large and rapidly evolving science. As progress is made in various topics of Complexity Theory, the individual topics will help to re-enforce each other. Our goal has been to reduce the requirement and dependence upon detailed a priori information about known attacks and detect novel attacks by computing vulnerability and detecting anomalous behavior based upon an inherent, fundamental property of information itself, namely, its complexity and sophistication. Results of complexity measures applied to network protocols, processes, and information have been presented and related to Information Assurance and network fault tolerance. Active networks form an ideal environment in which to study the effects of trade-offs in algorithmic and static information representation because an active packet consists of both code and static data. The code can contain the protocol or a compressed form of the data to be transported. If the code is the protocol, then information about the complexity of the protocol can be gleaned from the active packet code. An active packet that has been reduced to the length of the best estimate of the Kolmogorov Complexity of the information it transmits will be called the minimum size active packet. There are interesting relationships between Kolmogorov Complexity, prediction, compression and the model size used in the Active Virtual Network Management Prediction (AVNMP) mechanism. These relationships are throughout this report.

14. SUBJECT TERMS
Active Networks, Complexity Theory, Information Assurance, Kolmogorov

15. NUMBER OF PAGES
156

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Figures

Tables

Preface

A precise definition of complexity itself has been an ongoing debate, and there are many definitions; this report briefly compares and contrasts the definitions. It then settles upon algorithmic complexity, specifically, Kolmogorov Complexity, as a working definition and proceeds to explore the unique paradigm shift that appears when communication network fault tolerance is viewed through the lens of Kolmogorov Complexity. Our application of complexity to communication networking is directed toward both the current and next-generation Internet. Many emerging communication network technologies are discussed in this report such as self-healing networks, intelligent and predictive networks, active networks, predictive network management, and information assurance and network security technology. However a common thread, namely, the role of complexity, emerges to tie together these previously disparate technologies in new and unique ways. In addition, the application to information assurance is an extremely timely topic, given Microsoft's recently announced focus on fixing their product's security flaws. Recent tragic terrorist events clearly demonstrate that the civilian and military internets are vulnerable targets. We hope to spark new ideas in multidisciplinary fields; we are standing on the shoulders of two giants—the communication networking community and the founding fathers of algorithmic information theory—while attempting to make the work of both communities understandable to each other as well as to a general audience in the hope of synthesizing new ideas in the minds of all readers.

—Stephen F. Bush February 03, 2002

Acknowledgements

The work presented in this report was funded in full by DARPA, under the auspices of the Fault Tolerant Networking Program. Our thanks go to Doug Maughan, the Active Networks and Fault Tolerant Networking Program Manager, and Scott Shyne, Air Force Rome Labs, for their generous support.

Summary

A seminal contribution of this effort was presented—the development of a complexity-based information assurance metric for vulnerability analysis. The metric proposed is Kolmogorov Complexity. Advances in computable estimates of Kolmogorov Complexity are indicated, as well as additional applications of Kolmogorov Complexity for fault tolerant communications in general. Unless vulnerabilities can be identified and measured, the information assurance of a system can never be properly designed or guaranteed. An underlying definition of information security is hypothesized based upon the attacker and defender as reasoning entities, capable of learning to outwit one another. Estimates of Kolmogorov Complexity provide such an objective parameter with which to provide information assurance through anomaly detection and objective model development. The capability of this metric is limited in part by the accuracy of its estimation, which must be traded against computational expense. The Optimal Symbol Compression Ratio Algorithm, used to estimate complexity and sophistication, provides additional capability to discern anomalous behavior in information systems. Further research is needed to develop strategies for cost-effective use of this paradigm across entire systems.

The desirable properties of a metric for security are examined (Section 3.3). In order to further the development of a realistic metric, a general model for studying information assurance is proposed (Section 4). Next, a definition of vulnerability is proposed in terms of a new model based on Turing Machines (Hypothesis 4.1), and engineered properties of information assurance with an analogy to mechanical engineering are proposed in terms of the new model. The analogy with mechanical engineering is called Brittle Systems (Section 5) and involves the design of information assurance in a manner that accounts for tradeoffs in performance and degradation of information assurance in a system. Information assurance is also examined from the perspective of set theory and a topological space (Section 3.5). This is particularly relevant towards understanding the operation of the metric with regard to secure composition and the inherent limits of applying safeguards to a system.

The advantages and drawbacks of Kolmogorov Complexity are discussed, including its incomputable nature. However, computable estimates (Section 6.2) of Kolmogorov Complexity are explained, as well as additional useful applications of Kolmogorov Complexity for communications in general. These additional applications are important because they demonstrate how information assurance is an integral part of information system design. Next Theorems 6.1 and 6.2 concerning the conservation of complexity (Section 6.7) within an information system were discussed. This led to a Swarm experiment that monitors the evolution of complexity in a dynamic and complex system and examines our ability to monitor the complexity as it evolves. Unless vulnerabilities can be identified and measured, the information assurance of a system can never be properly designed or guaranteed. Results from a study on complexity evolving within an information system using Mathematica (Section 9.2), Swarm, and a new Java complexity probe toolkit (Section 9.4), developed by this project, were presented in this report. An underlying definition of information security was hypothesized (Hypothesis 9.1) based upon the attacker and defender as reasoning entities, capable of learning to outwit one another. This leads to a study of the evolution of complexity in an information system and the effects of the environment upon the evolution of complexity. Understanding the evolution of complexity in a system enables a better understanding of how to measure and quantify the vulnerability of a system. Finally, the design of the Java complexity probes toolkit under construction for automated measurement of information assurance is presented (Section 9.5). A dialog is included that contains typical questions about the relationship between complexity and information assurance. This dialog is best read after reading the introduction Kolmogorov Complexity (Section 6.1) or for someone already familiar with complexity theory who wants a quick overview of the approach taken on this project toward the relationship between complexity and information assurance.

Another result of this project is the concept of conservation in the evolution of complexity in a system and the search for a bound on the change in complexity such that abnormal behavior can be detected when the bound is exceeded. This work demonstrated a promising approach for further

exploration into the laws governing complexity and the evolution of complexity within a system using simulation. Finally, complexity probes were developed to enhance a security-engineering tool based upon an electrical engineering paradigm with complexity as the resistance to insecurity flow.

Blindly applying current communication and computation technology on MEMS devices would be fighting a losing battle against nature. The proposition this report hoped to reinforce in the readers' mind was that MEMS devices can be more efficiently engineered by working with, instead of against, the environment in which they are placed. Specifically, two approaches were proposed for revolutionary gains in MEMS device communication. The first was to view all network devices as computational or active devices. Computation can take many forms. The amount of computation may vary, but every device has some type of computation, either programmed or ambient. Use of computation in an optimal manner is the same challenge faced by active networks. Thus, advances in active networks and networks of MEMS devices are mutually beneficial. The second approach was to optimize networks of MEMS devices via exploiting emergence. Understanding emergence requires understanding complexity; that relationship was touched upon relative to networking in this report. Use of emergence shows promise as a means to precisely engineer desired characteristics into systems of MEMS devices resulting in reduced size by removing unnecessary computation and control.

This report shows that a genetic algorithm shows sudden decreases in complexity of the population between generations as the algorithm evolves in response to the fitness function. Lower complexity corresponds to greater homogeneity in the population and greater fitness to the chosen criterion. Thus it can be clearly seen that complexity can be used as one indicator of progress in evolution of the genetic algorithm. A framework for testing the injection of fitness functions into an active network that evolves solutions via a genetic programming technique has been implemented. Future work involves testing the response time to heal and the resiliency of the network in the presence of faults.

Introduction

Complexity is being studied in a myriad of disciplines and in many different ways. Measures of complexity have been derived in attempt to understand complexity, but they have disadvantages, either they do not fully capture the nature of complexity, or they are fundamentally incomputable.

INTRODUCTION TO COMPLEXITY

The word complexity comes to us from the Latin word *complexus*, meaning 'in totality' or 'a whole set consisting of many interconnecting parts.' The definition of the word includes the connotation 'difficult to understand'. The Universe, as well as every object in the Universe, consists of many interconnected parts. Humans have attempted to reduce the apparent complexity of nature, in other words to understand the Universe, by observing particular instances of the operation of subsets of interconnected, or interacting, parts. Hypotheses are generated, experiments to test those hypotheses are developed, and the outcome of the experiments either reinforces, or counters the hypotheses. Variations on the original hypotheses, or entirely new hypotheses are generated and tested and the cycle continues. Science has progressed in this manner in search of the essence, or most general underlying explanation for as many phenomena as possible. As discussed later in this report, the very critical act of hypothesis formation and testing, the Scientific Method, the foundation of science upon which Man depends for advances in every aspect of civilization, is itself governed and characterized by complexity.

How far can the Scientific Method, described above, take us in understanding Complexity Theory? When one studies complexity as a science, the focus becomes a simplified understanding of large numbers of interactions. Subtle, yet insidious problems render the study of complexity a challenging problem. The particular details of individual parts, important in specifying interactions, are less important to the understanding of complexity than the interactions. An exception is when the parts themselves consist of many interacting parts. This implies the existence of layers of complexity. How can one obtain a perfectly closed subset of the Universe in which to test hypotheses concerning complexity? The mere act of measuring any characteristic of such a system violates closure. How can one be certain that there are no interactions at some unknown level with in the supposedly closed system and the rest of the Universe? Is it possible for one system to measure the complexity of another more complexity system? These are some of the questions that we will explore in this report. Gödel has demonstrated that a system cannot completely describe itself with perfect fidelity. Berryman's Paradox suggests that any algorithm capable of computing complexity must be at least as complex as the object whose complexity is being measured. How can one measure the complexity of an algorithm that measures complexity without requiring a more complex algorithm? Measuring the complexity of the more complex algorithm requires an even more complex algorithm, ad infinitum.

While studying Complexity Theory, researchers focused on the science of interactions of large numbers of parts, have noticed that something amazing happens under certain conditions. Unexpectedly complex results, based upon simple interactions can occur. This is known as emergence. Detecting and controlling emergence could also lead to groundbreaking results. The implications are that programming simple interactions while letting emergent behavior handle the bulk of the work in a robust manner could control the desired characteristics of a system. In other words, complexity theory, specifically through emergence, could provide a new and much more efficient and robust form of control. Ultimately, complexity theory and emergence could progress to self-organizing systems. These are systems whose natural tendency is to align in a form optimal to the task required. An example is a self-healing system, that is, a system that inherently reforms to mitigate a fault.

MEASURES OF COMPLEXITY

There have been many attempts to define and measure complexity. Attempts to define the complexity of a system might be broadly described as attempts to remove portions or patterns, of the system that are simple, leaving behind the portions that are complex. The size of the remaining portions of the system must then contain the complexity of the sys-

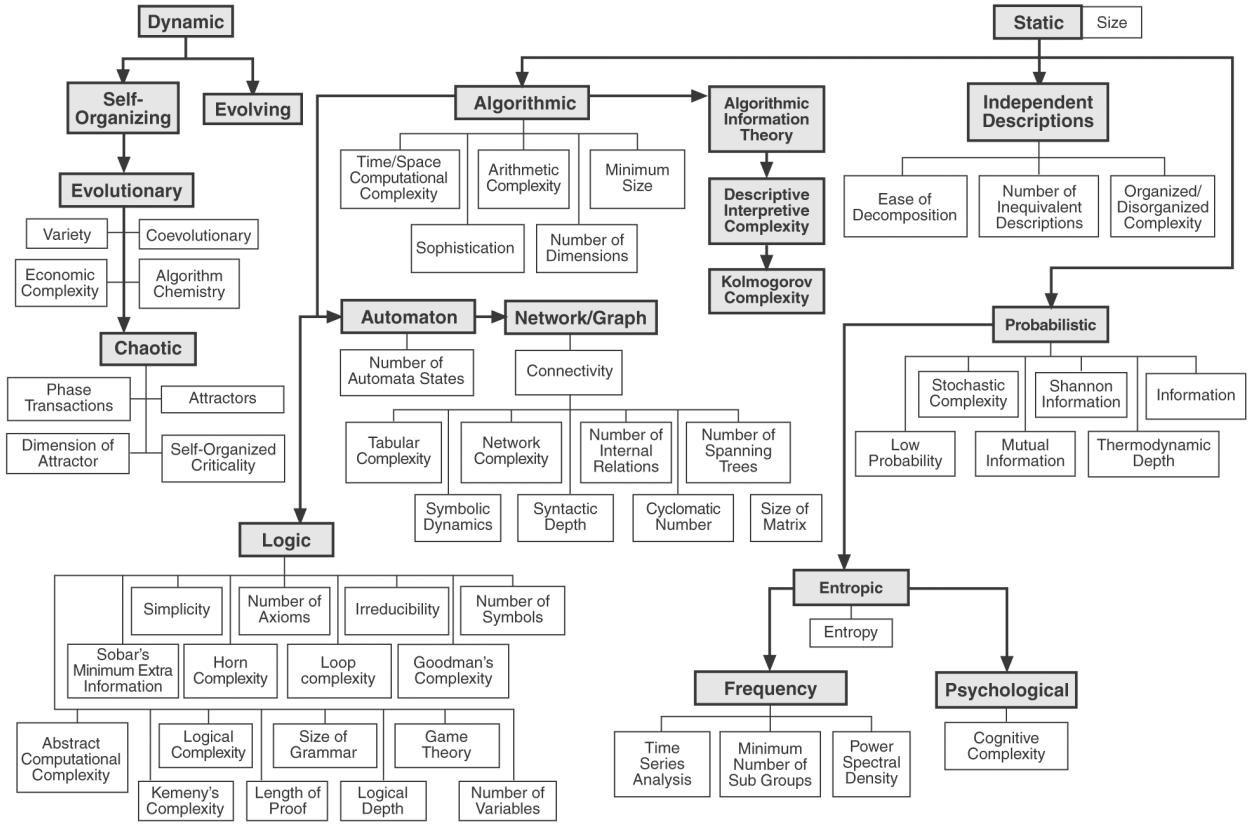

Figure 1. Definitions and Measurements of Complexity.

tem. In Figure 1 an attempt is made to cluster selected known complexity measures into categories. The purpose of this figure is to show the great variety of complexity techniques and to be able discuss broad classes of techniques. Except for a few specific exceptions, details of each and every technique will not be discussed. The categories in this classification describe the complexity estimation technique and thus suitability to types of systems. The arrows indicate subclasses of complexity estimation techniques. The highest-level classifications are Static and Dynamic techniques. Static techniques assume the system whose complexity is to be estimated does not vary with time as the estimation calculation executes. A snapshot of a Dynamic system is also considered a static system. Dynamic techniques allow the system to change with time; in fact some require that the system under analysis change with time.

The Static complexity estimation techniques can be further sub-classified into Algorithmic, Independent Descriptions, and probabilistic categories. Algorithmic techniques attempt to use an algorithm as a fundamental description of the complexity of a static snapshot of a system. Independent Description techniques attempt to count the number of irreducible components needed represent a system, such as number of dimensions, number of independent models, or number of irreducible components. Probabilistic techniques generally assume that low complexity components are more likely than higher complexity components. Probabilistic techniques also assume attempt to use probability to determine independence of sub components of a system allowing the system to be partitioned into independent components. In this work, Algorithmic techniques are most relevant because of the nature of computation in the form of executable algorithms and its relation to the transmission of information in the form of static data. Algorithmic techniques are further sub classified into Propositional Logic-based and Automata-based techniques. Algorithmic techniques also include minimum description methods that seek to estimate complexity by determining the smallest size to which a description can be compacted. Complex descriptions, containing a larger

number of "random" interactions and lacking any form of repeatable patterns, cannot be described as compactly as simple descriptions. Kolmogorov Complexity is a complexity measure that falls within this category.

Vladimir Gudkov has been exploring the minimum number of dimensions required to characterize network information flow [163–166]. Vladimir's work could be categorized in the Dynamic, Self-Organizing, Evolutionary, Chaotic group of complexity theory techniques. Vladimir's approaches the development of network behavior description in terms of numerical time-dependant functions of protocol parameters. This provides a basis for application of methods of mathematical and theoretical physics for information flow analysis on network and for extraction of patterns of typical network behavior. The information traffic can be described as a trajectory in multi-dimensional parameter-time space with dimension about 10–12. The result of his work could help to improve our Kolmogorov Complexity estimators.

Kolmogorov Complexity is a measure of descriptive complexity contained in an object. It refers to the minimum length of a program such that a universal computer can generate a specific sequence. A good introduction to Kolmogorov Complexity is contained in [118] with a solid treatment in [10]. Kolmogorov Complexity is related to Shannon entropy, in that the expected value of K(x) for a random sequence is approximately the entropy of the source distribution for the process generating the sequence. However, Kolmogorov Complexity differs from entropy in that it relates to the specific string being considered rather than the source distribution.

The major difficulty with Kolmogorov Complexity is that it is not computable. Any program that produces a given string is an upper bound on the Kolmogorov Complexity for this string, but you can't compute the lower bound, yet as will be discussed later in this section, estimates have shown to be useful in providing information assurance and intrusion detection.

Kolmogorov Complexity is a measure of descriptive complexity that refers to the minimum length of a program such that a universal computer can generate a specific sequence. Universal computers can be equated through programs of constant length; thus a mapping can be made between universal computers of different types. The string x may be either data or the description of a process in an actual system. Unless otherwise specified, consider x to be the program for a Turing Machine described in Definition 1.

$$K_{\varphi}(x) = \left\{ \min_{\varphi(p) = x} l(p) \right\} \tag{1}$$

$$K_{\varphi}(x|y) = \left\{ \begin{array}{c} \min_{\varphi(p,x) = y} l(p) \\ \\ \infty, \textit{ if there is no } p \textit{ such that } \varphi(p,x) = y \end{array} \right\} \tag{2}$$

Conditional Complexity, described in Equation 2, quantifies the complexity of string *x*, given string *y*. Intuitively, it is the additional complexity of string *x* beyond that in string *y*. This definition of Kolmogorov Complexity is used repeatedly throughout the remainder of this report.

Kolmgorov Complexity has been shown to provide a useful framework from which to study objective metrics and methodologies for achieving information assurance. Recent results have shown promise for complexity estimators to detect FTP exploitsand DDoS attacks. Complexity is attractive as a metric for information assurance because it is an objective means of characterizing and modeling data and information processes for the purpose of benchmarking normal healthy behavior, identifying weaknesses, and detecting deviations and anomalies.

Since exact measurement of Kolmogorov Complexity is not computable, estimators are required. The accuracy and computational requirements of estimators together determine the capability or practicality of use for a given application. For example, the very crude complexity estimate of empirical entropy carries very little overhead, but is suitable for some applications. Other applications can benefit from complexity metrics when more expensive estimation algorithms are utilized, but the computational expense may not be feasible.

In this report we motivate the use of complexity metrics for information assurance by discussing several applications of complexity metrics for information assurance, each of which depends in some sense on accurate complexity estimators. We then discuss and compare several ubiquitous complexity estimators, their accuracy, and computational expense. Finally, we introduce a new complexity estimator and benchmark its capability against others for the FTP exploit detection application.

Methods, Assumptions and Procedures

1. Methods and Assumptions

1.1 METHODS AND ASSUMPTIONS

The project report is presented in the form of three hypotheses that correspond with the Imperishable Network statement of work. Following each hypothesis is a list of accomplishments relating to validation of that hypothesis.

Hypothesis 1

The first project goal is to explore the hypothesis that information comprised of observations from an event, such as a fault or attack, which is generated by a single root cause, is highly correlated. Highly correlated data has a low complexity and a high compression ratio. However, this project is not focusing on legacy static compression algorithms, but rather algorithmic compression. Algorithmic compression involves code that can dynamically change, and when executed, regenerates the intended data. The compression code is designed to be a hypothesis about the data to be compressed. The more accurate the hypothesis, the more efficient the compression. Algorithmic compression and prediction are tightly linked; if a program can predict data (generating more data than the program's size), then it is, by definition, an algorithmically compressed form of the data. Active networks form an ideal vehicle for transmitting this form of fault information because they facilitate the transmission of code within the network. The most highly compressed, and thus most likely, fault representations are transmitted faster and farther due to their smaller size. Kolmogorov Complexity, *K(x)*, measures the size of the smallest program capable of representing a particular piece of data, thus providing guidance as to the optimal amount of information within an active packet to be in the form of code versus data. Specific accomplishments towards this goal are:

1. A new algorithm that incorporates both Kolmogorov Complexity and entropy, facilitating our study of the relationship between them, has been devised to both estimate complexity and perform compression.
2. A DDoS attack detection algorithm, based upon our hypotheses regarding complexity theory, has been implemented. Testing is underway within our Active Network testbed. The algorithm makes use of a fundamental theorem of Kolmogorov Complexity we derived that states: For any two strings *X* and *Y*, $K(X,Y) <= K(X) + K(Y)$, where *K(X)* and *K(Y)* are the complexities of the respective strings and *K(X,Y)* is the joint complexity of the two strings. Stated more simply the joint Kolmogorov complexity of two strings is less than or equal to the sum of the complexities of the individual strings. In other words, the joint complexity of the string (data stream) decreases as the correlation within the string increases. This property is exploited to distinguish between concerted denial-of-service attacks and cases of traffic overload. The assumption is that an attacker performs an attack using large numbers of correlated packets generated from different locations but intended for the same destination. Thus, there is a lot of similarity in the traffic pattern. A Kolmogorov complexity based detection algorithm can quickly identify such patterns. On the other hand, a case of traffic overload in the network tends to have many different traffic types and the traffic flows are thus highly uncorrelated, appearing to be "random." Our algorithm samples distinct packet flows (distinguished by their source and destination addresses) to determine if there is a large amount of correlation between the packets. If it is determined to be so, then all suspicious flows at the node are again correlated with each other to determine that it is indeed an attack and not a case of a traffic overload. We compared our technique to a simple packet counting algorithm for DDoS detection and found that our technique is much more sensitive in detecting attack. Complexity differential is defined as the difference between the cumulative complexities of individual packets and the total complexity computes when those packets are concatenated to form a single packet. In effect, we use the measure of the compressibility of the packets accumulated in a given time interval to determine correlation. We believe that it will also be much more accurate in separating false alarms from true attacks. This set of experiments is underway.

Hypothesis 2

It is hypothesized that the degree to which information can be compressed algorithmically is a measure

of the *ease of understanding* the information, particularly by an attacker. This concept is used to estimate the vulnerability of a system through given observable points within the system. Apparent complexity has been defined in this project as the complexity normalized to the prior knowledge of an individual. Prior knowledge can be obtained automatically by watching a potential attacker within a fishbowl, during an attack, or assigned by other means. We believe that our technique is more efficient, robust, and ubiquitously applicable than developing a database of vulnerabilities and testing for all potential vulnerabilities as most people have been attempting.

1. Complexity probes (in the form of Magician Active Packets) have been developed and a test plan put together that would help verify this hypothesis.

Hypothesis 3

It is hypothesized that the algorithmically compressed representation of a network fault provides unique opportunities for *seeding* the composition of solutions that mitigate the fault. Thus, the relationship between complexity and fault/solution composition is being explored for the development of self-organizing solutions.

1. A simple Mathematica 'simulation of an active network has been developed that focuses on the algorithmic aspects of data encoded within a packet. It abstracts away networking details allowing a focused study of the tradeoff in algorithmic versus static information transmission.
2. A Mathematica'-based Genetic Algorithm simulator has been instrumented with complexity estimation. A decrease in complexity was noted during the initial evolutionary stages of all genetic algorithms tested. In other words, as optimal solutions evolved, the measurement of the complexity of the total system decreased.
3. Previous work using genetic algorithms to both determine Kolmogorov Complexity and generate algorithmic representation of multimedia data were recently found in the literature and help to validate our approach.

1.2 PROJECT CHALLENGES

Accurate estimation of Kolmogorov Complexity is salient to its ubiquitous application to network fault tolerance and security. We will benchmark our new compression algorithm and estimator for K(x) against other means in search of a model base under MML that is effective in efficiently and accurately estimating Kolmogorov Complexity.

With respect to the DDoS Kolmogorov Complexity application, its performance will be compared to other detection algorithms that are currently in use. In particular, its performance has to be measured in terms of resource tradeoffs, detection and false-alarm probability and response time. It is hypothesized that other DDoS detection techniques, while optimized for detecting certain types of attacks, will not be as robust in detecting all types of attacks.

A challenge for this project is identifying or developing the fundamental theory for composition of solutions using Kolmogorov Complexity.

1.3 CHALLENGE QUESTIONS

1. How well can Kolmogorov Complexity be estimated?
2. What are the benefits and tradeoffs associated with algorithmic information transmission?
3. Can a network fault be represented algorithmically?
4. Can the algorithmic representation of a network fault seed the formation of optimal solutions?
5. What is the meaning of the minimal Turing Machine generated from a compression algorithm?
6. Can a tolerance be incorporated to tradeoff computation and complexity estimation?
7. Is there a convergent form of complexity estimation, i.e. allowing the complexity to converge to a value?
8. Can information fusion be accomplished more efficiently using algorithmic forms of information?
9. Could information that is in-transit within the network be combined so as to reduce complexity? Example: Bioinformatic data/algorithms could be fused within the network from multiple sources and only those combinations leading to lowest complexity are kept. Assuming that lowest complexity indicates most likely explanation. Network complexity reduction: think of complexity as energy; the network tries to find lowest energy state.
10. What are the fundamental theorems derivable from Kolmogorov Complexity that can allow us to define algorithms for self-composition?
11. Much work has been done in the past on detecting and measuring the impact of faults. How can one quantify and measure the impact of solu-

tions, which are of equal importance in matching (composing) faults and solutions?

12. Chaos Theory views system operation in terms of phase (state) space. Attractors are *patterns* in phase space in which the system tends to remain (a coherent organization). Think of attractors as the gravitational pull keeping the network together (or functioning properly). Because attractors are patterns, they are highly compressible (low complexity). Is a measure of the system to self-organize is a ratio of the size of the attractor versus the size of the operational phase space?
13. Consider algorithmic representation of faults. Using reversible code (anti-code) it would be possible to reverse the computation, i.e. eliminate the fault. I believe they have developed anti-code compilers. What could one say about the complexity of code and its corresponding anti-code?

2. Discussion

A key assumption of this work is that complexity and information assurances are related. Clearly, the system should appear complex to an attacker and simple to a legitimate user. However, in order to anticipate questions that an astute reader might have, this section is written in the form of a question and answer dialog. Our colleagues have raised some of these questions; however, identities will not be revealed in order to protect the innocent.

Question: Aren't higher complexity systems more vulnerable to attack? How does that correlate with an attacker following the path of least vulnerability?

Answer: Assume the Kolmogorov Complexity of a system can be represented by x, that is, $K(x)$, is by definition, the size of the smallest program capable of generating x. Thus, $K(x)$ does not vary with the implementation of the system. In Section 5.0, we discussed how the brittleness of a system changes with respect to the efficiency of the implementation. We define $AK_r(x)$ to be the complexity of system x as viewed by attacker r. While it is intuitively and empirically true that more complex systems tend to have more security problems this is attributed to the inability of the defender to understand their own system fully and thus comprehend how to best defend it. It is the differential between the defenders understanding of a system and an attackers understanding that is a measure of true information assurance. The desired goal is simplicity for the defender and complexity for the attacker.

Question: Suppose, as an attacker, I have found an encryption key. The encrypted data appears very complex, yet knowing the key, I can easily obtain the information.

Answer: There is an estimate of complexity known as Minimum Data Length description that involves compressing both data and the hypothesis used to generate that data. If the apparent complexity, $AK_r(x)$, is estimated for an attacker known to have the encryption key, then the complexity will be very low. The complexity of the encrypted data is always $K(x)$. The apparent complexity of the data in the absence of the encryption key is much greater than $K(x)$. When an attacker gains the key the differential between $K(x)$ and apparent complexity is dissolved and all security is lost.

Question: Wouldn't an attacker choose to hide inside a more complex component than a simple one?"

Answer: A similar question appears in the study of work factor as an information assurance metric. An attacker may be willing to spend more effort, or take a higher complexity path, if he has the time and has a suitably high interest in avoiding detection. Again, the attacker can exploit the defenders inability to understand their system.

Question: Wouldn't estimating the complexity of every bit-stream in a system require a lot of overhead? Is there a more efficient way?"

Answer: An implementation of complexity-based vulnerability analysis that requires bit-stream level computation for every possible data flow would require a lot of overhead. One possible approach is to look at more aggregate views of the system and determine complexity from SNMP variables as an example. However, consider that non-complexity-based alternative approaches that attempt to include extreme detail quickly find the problem to be overwhelming and in addition, such approaches are generally easily broken if they miss a particular detail.

Question: One of your estimates of complexity relies upon the inverse compression ratio. Isn't that based upon entropy rather than complexity?"

Answer: Yes, our initial complexity estimation technique relied on the inverse compression ratio. This was chosen as a Kolmogorov estimator because it appeared to be a low overhead and easy to implement technique.

Question: Since there are such sloppy bounds associated with any estimate of K(x) isn't the ability to measure and utilize conservation of complexity a pipe dream?"

Answer: Estimating Kolmogorov complexity is a challenge. Our current research is focused on finding the best metrics and quantifying bounds associated with these metrics to determine the usability of conservation of complexity to solve real problems. Our hypothesis is that beyond certain thresholds abnormal behavior will be noticeable using conservation of complexity.

2.1 SURVEY OF RELEVANT EXISTING SECURITY TECHNIQUES AND THEORY

A fundamental basis for information security is elusive. Numerous theories have come to light lately that look for a fundamental basis for the study of security systems. In this chapter we review some of the more fundamental work in this area. We conclude this chapter with two analogies—to Thermodynamics and to Electrical Engineering—that yield an intuitively pleasing basis that we would like to explore.

Bennett/Zurick

Physics of information has been studied for decades with many interesting theoretical contributions by Zurek et al.[117] and Bennet et al [99]. This body of work identifies Kolmogorov Complexity as a basic property inherent in the physics of information, and strives to resolve actual physical laws of energy with information laws. Quantum computing is a related area. Our work applies many of the concepts introduced by Bennet and Zurick into the Information Security domain.

Harmon

Reference [24] describes a recently developed model of information systems where the fundamental devices are processors, routers, memory components and communication components that serve to affect information in the form of modulated energy. Assumptions and postulates are well laid out to provide possible future experimental validation of this model. System complexity is defined as the number of dependencies that exist between pieces of information. Our approach differs from this approach in that we will use the fundamental quantity of Kolmogorov Complexity as our basic building block.

Fisher Information

Reference [125] puts forth a unification of the laws of physics and the statistical quantity known as Fisher Information. This shares with our approach the gravitation towards a fundamental parameter, in this case Fisher Information, which applies locally to specific data as opposed to general source distributions from which data are generated, as is the case with Shannon entropy. Fisher information is defined as follows:, where Λ is the likelihood function described by: given Z, a set of observations and x, a time invariant parameter measured by observation set Z. A resolution of our approach to these results is desired.

Current Security Techniques

Information security (or lack thereof) is too often dealt with after security has been lost. Back doors are opened, Trojan horses are placed, passwords are guessed and firewalls are pierced—in general, security is lost as barriers to hostile attackers are breached and one is put in the undesirable position of detecting and patching holes. In fact many holes go undetected. Breaches in other complex systems that people care about are not handled in such an inept manner. Thermodynamic systems, for example, can be assured of their integrity by the pressure, heat or mass the system contains. Hydrostatic tests can be performed to ensure that there are no "holes," and the general health of the system can be ascertained by measuring certain parameters. A problem is identified as soon as the temperature or pressure drops, and immediately one can take action to both correct the problem and to isolate other areas of the system from harm. But how does one perform a hydrostatic test of an information system? What conserved parameters exist to measure the health or vulnerability of the system? How can one couple the daunting task of providing a system where vulnerabilities are readily measurable with the required need for simplicity of use for authorized users? We explore these issues through various analogies and propose that only through monitoring objective quantities inherently related to information itself can the science of information assurance move beyond patching holes.

Analogy to Thermodynamics

An attractive analogy for an information security system is given in an analogy to thermodynamics. In thermodynamic systems laws of conservation and energy flow allow monitoring of the health of the system through parameters such as temperature, heat, and volume. One does not, for example, in a thermodynamic system, wait for all the heat to drain from a heat exchanger and a rat to come inside to announce that there is a problem. One can tell from parameters such as temperature and pressure that the system is behaving abnormally. Concepts such as entropy and mass map nicely to the information security domain. Through our exploration of Kolmogorov Complexity, we pursue the analogy to thermodynamics.

Analogy to electrical engineering

Analogies have been drawn between basic electrical engineering parameters, such as impedance, cur-

rent, and voltage, and infor mation assurance of an information system. Electrical current could be compared to information flow to dishonest participants. High resistance or good insulation on an electrical cable could represent a network with few holes. Through sufficient mappings, some of which are shown in Table 1, simplified models, parallels to Norton and Thevenin equivalent circuits, could be developed through simple measurements and transformations. Additionally, the large body of work dedicated to detecting and protecting against electrical faults and disturbances could provide some benefit to the field of information assurance.

Table 1 Electrical and information assurance properties

Electrical property	Security property
Current	Data flow to or from dishonest participants
Voltage	Information assurance potential
Resistance	Resistance to data flow to dishonest participants
Inductance and capacitance	These values follow by direct insertion of above analogies into electrical definitions of inductance and capacitance.

Here the information assurance potential is a measure of the ability of a system to defend or a perpetrator to intrude upon an information system. The equivalent resistance of the system is determined by considering and quantifying all of the possible systems (Figure 1). A distinction is made between active networks [3] and today's legacy, or passive networks, in this proposed electrical engineering paradigm. The work involved in forwarding a packet, whether active or passive is current that can cause a node to do work, that is, current in a motor winding that causes energy transfer in a different form. With regard to active packets and information theory, passive data is simple Shannon compressed data, and active packets are combination data and programs whose efficiency can be estimated through Kolmogorov Complexity. Information assurance laws must be able to deal with many alternative representations of information. Section 3 discusses an electrical engineering grid type of information assurance tool.

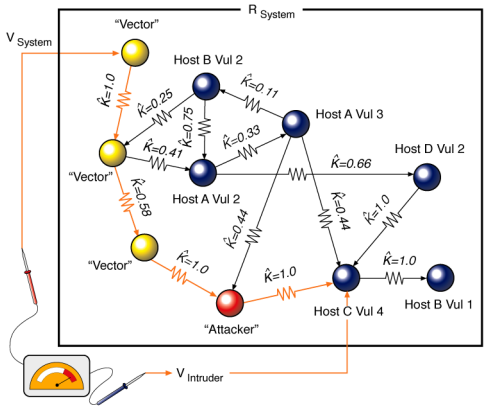

Figure 1. Electrical and Information Assurance Properties.

2.2 THE NETWORK INSECURITY PATH ANALYSIS TOOL (NIPAT)

Consider a specific grid-based information assurance tool known as the Network Insecurity Path Analysis Tool (NIPAT). NIPAT is a powerful security analysis tool developed at GE Global Research that has been improved by the results of this project in complexity-based vulnerability analysis. NIPAT serves as a positive representation for grid-based information assurance tools in general. Section 3.1 discusses their weaknesses. Figure 2 displays 2,000 vulnerabilities found on a few nodes of a network that were thought to be reasonably secure. Vulnerabilities are displayed in Figure 2 by host and type. The number along each edge of the graph repre-

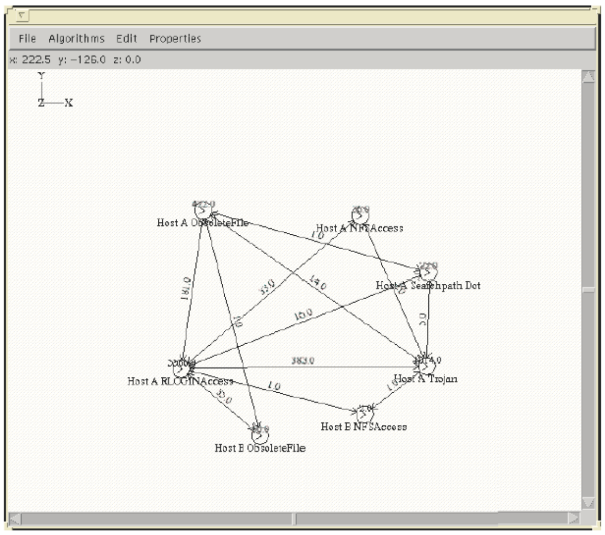

Figure 2. A Grid-Based Tool in Action.

sents the number of opportunities available to the attacker to reach the next vulnerability. NIPAT can automatically generate a directed graph representing the security vulnerabilities of a network. This information is gathered from network security software agents. The security vulnerability graph for a typical network can be extremely dense; however, the object-oriented nature of the security model is useful in choosing the level of abstraction required. For example, it may be possible to display the vulnerability graph for Unix hosts in general and to hide the details of individual Unix variants. NIPAT determines the degree to which specified targets within the network can be compromised. The vulnerability chain is displayed as a directed graph. Nodes represent vulnerabilities whose security may be compromised, and edges represent paths from vulnerability to vulnerability. The larger the value of the edge label, the greater the vulnerability. The focus of this effort is on the mathematical representation of information assurance; thus, the underlying database and data gathering agents are not discussed in detail here. See Appendix C for more on the operation of the NIPAT tool.

The information assurance model assumed by NIPAT is that of an attacker who has a finite amount of resources with which to penetrate network security. A resource vector for the attacker is assumed. The cost to the attacker of using each of the resources against a particular network is defined by consumption functions. The cost to network security of implementing security measures is defined by a k-dimensional security function. The attacker's resource vector consists of the strength of each element of the attacker's resources. For example, the password decryption resource value would consist of the attacker's CPU speed and amount of time the attacker would be willing to spend on the attack. The NFS spoofing resource value would be the time to install and run the NFS spoofing software multiplied by the probability that the attacker has access to such software. The host spoofing resource value would be a function of the attacker's ability to evade the physical security of a network and install or modify a host IP address. The consumption function vector is the complement of the attacker's resource vector. For example, a network with good password encryption algorithms or whose users use well-chosen passwords will have a high value for the consumption function for password decryption. Clearly, attempting to define all possible security threats to any system is a huge undertaking. However, the scope of network security is confined to the security object model. In this example, the following techniques are assumed to be available to the attacker: password decryption, NFS spoofing, hosts spoofing, and application security faults. Password decryption assumes the attacker has a program capable of decrypting users' passwords. NFS spoofing involves violating security to mount another user's file system; host spoofing is causing a host to appear to the network as a different host; and an application security fault is taking advantage of an application-programming fault in order to attack the security of a system. Each of these resources is measured in units of time. Thus, an attacker with a powerful computer and a willingness to wait a long period of time to break into a network will have a large password decryption resource. An attacker with physical access to the network and competent knowledge will have a high host spoofing resource value because such an attacker can physically connect a host to the network. An attacker with much experience and knowledge of applications will have a large application security fault resource.

Another form of vulnerability analysis involves detecting vulnerabilities that change over time. The network monitoring tool quantifies the vulnerability of a system in terms of percent of patches which fail to have the correct signature, percent of files which are accessible to others besides the owner, and percent of passwords which can be guessed with a given password generation tool. Clearly, vulnerability checks such as these increase the security of the network. Both the type of information gathered and the frequency with which the information is updated quantify the effectiveness of a network monitoring strategy. If the information is not updated frequently enough, an attacker may have penetrated network security and left before network security is aware of the situation. An estimate of the effectiveness of the monitoring system is based on a profile of network security attacks on the Internet and the following parameters: time to monitor patches, Trojan horses, passwords, and any other vulnerabilities. The attack rate is assumed to be Poisson. The average attack rate, based on Internet incident reports from an anonymous site for a six-year period, is five attacks per month. Also the Defense Information Systems Agency has determined by experimental means [107] that only 0.7% of incidents are actually reported. Thus, for each path in the network secu-

rity vulnerability chain, the cost to the attacker is the probability of being detected multiplied by the cost function that the additional monitoring provides.

The following list describes the capabilities and benefits that a vulnerability assessment tool could provide: An automated network security assessment tool should have the capability of automatically generating a directed graph of vulnerabilities. This information can be gathered from network security software agents. The security vulnerability graph for a typical network can be extremely dense; however, the object-oriented nature of the security model may be useful in choosing the level of abstraction required. For example, it may be possible to display the vulnerability graph for Unix hosts in general and to hide the details of individual Unix variants. The network security vulnerability tool determines the degree to which the network security can be compromised based on minimizing an objective function that represents the cost to the attacker. Based on the vulnerability graph, the optimal deployment location and capability mix of the security agents is determined. Note that this is closed feedback loop; the security agents are sending information to the security analysis tool, which controls the deployment of the agents. The network security vulnerability assessment tool should indicate the degradation in the quality of service to legitimate network users as security counter measures are taken as well as dynamically indicate the security vulnerability of the network. An object-oriented prototype network vulnerability analysis tool, NIPAT, which implements most of the above requirements, has been implemented using Java. The vulnerability chain is displayed as a directed graph. Nodes represent entities whose security may be compromised, and paths represent the vulnerability of an entity. The larger the value of the path label, the greater the vulnerability.

Mathematica [139] provides an ideal environment for experimenting with symbolic mathematical concepts. The adjacency matrix, which represents the vulnerability graph from NIPAT, can be read into Mathematica. The directed, weighted adjacency matrix is used to determine the shortest path between every two nodes. The insecurity values can be displayed as a contour map and a density plot, where high areas in the topological view are secure, and those lower are relatively less secure, as shown in Figure 3 and Figure 4 for the system shown in Figure 2.

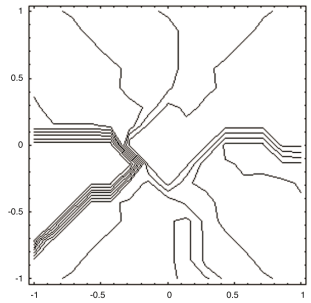

Figure 3. Topographical Map of Security.

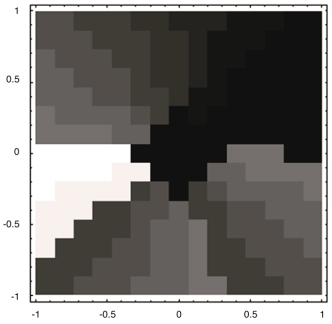

Figure 4. Density Graph of Security.

As has already been mentioned, extant forms of automated security vulnerability analyses rely on polling, which becomes infeasible in large-scale networks and in highly dynamic environments. Other approaches towards vulnerability analysis and intrusion detection need to be developed. There are two independent research efforts that are leading towards a mutually beneficial solution to the vulnerability assessment and network security problem. These research efforts are the human biological immune system approach to network security and Active Networks [3]. Active networking implements the cliché that *"the network is the computer."* Active networking allows users of the computer communications network to inject programs into the network to customize processing of user and application specific data. Thus, just as hormones control and regulate biological systems, active networks allow programs to travel the network modifying security behavior. The biological analog of intrusion detection is highly distributed. The advantage of a distributed intrusion detection system is that the probability of detecting an intruder increases significantly as the intruder is forced to pass through more independently operated intrusion detection systems. Biologically inspired forms of vulnerability quantifi-

cation would include injecting a network with a harmless virus and measuring how far it can spread throughout the network. This would clearly indicate the location of vulnerabilities. A more aggressive solution would involve "growing" many simple cells (processes) in a closed computer environment. These cells (processes) constantly mutate, reproduce when they successfully attack an intruder (non-self), and die when they attack legitimate system and user processes (self). Over time, by natural selection, only useful processes will remain which can be injected into a network and used to detect and attack intruders. Clearly, active networking enables new and more flexible security safeguards in addition to facilitating the development of the immunological approach towards network security.

A network security analyst can allocate security safeguards in order to minimize the entire network vulnerability, or to minimize the vulnerability from known attack points to particular targets. A quick study using NIPAT is presented. First, from a fundamental network vulnerability flow viewpoint, the strategy of allocating safeguards in combinations of serial and parallel strategies can be examined. Figure shows NIPAT analyzing an attack from host A

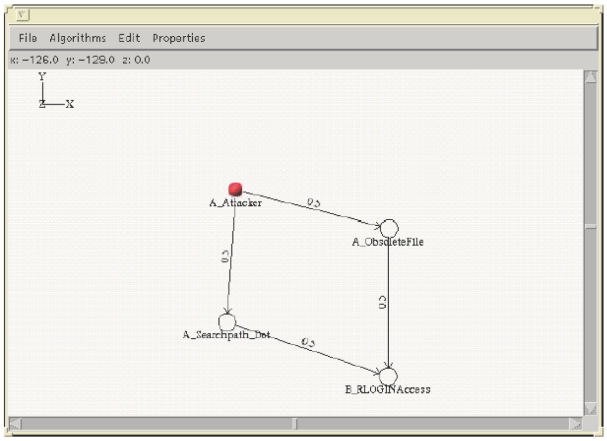

Figure 5. An Example of Security Safe Guard Assumption.

to host B. In this case, the number of opportunities has been normalized into probabilities. Figure 6 shows the results as security safeguards are removed. The solid line is the vulnerability of a single connection from the attacker to the defender having the same vulnerability flow as the links shown in Figure . With a probability of less than 0.6 a diversity of vulnerability types helps to increase security, but interestingly, above 0.6 it does not.

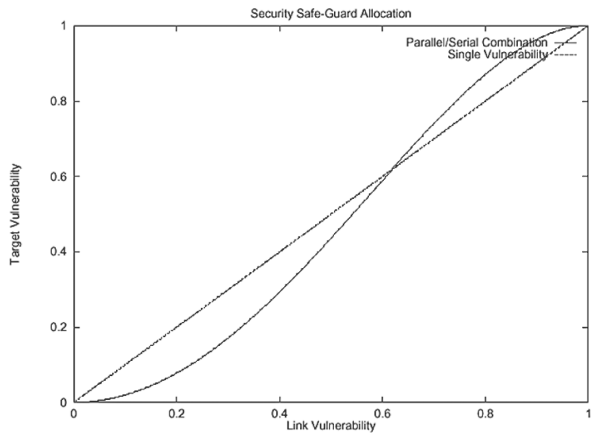

Figure 6. Series versus Parallel Vulnerability Attack.

Let us assume that vulnerability has been calculated by NIPAT to be either the maximum insecurity flow or probability of successful attack, where S represents security safeguards, $C(S)$ is the cost of security, and L is the cost constraint or some other hard resource limit. Next we discuss the cost in terms of impact on users; here it is strictly a financial cost or other resource constraint. Objective Function 3.1 shows how the optimal security safeguard allocations can be determined.

It is possible to use NIPAT to study various strategies of both defensive and offensive players in a network attack. Once an attack has been detected, the network command and control center can respond to the attack by repositioning security safeguards and by modifying services used by the attacker. However, cutting-off services to the attacker also impacts legitimate network users, and a careful balance must be maintained between minimizing the threat from the attack and maximizing service to customers. For example, various stages of an attack are shown in NIPAT in Figure 7 along the yellow path. Since the allocation of security resources never changes throughout the attack, the vulnerability of the target increases significantly with each step of the attack.

Our proposed enhancement would be to incorporate the following algorithm into NIPAT. Let CS represent the network service to customers, with a minimum accepted quality, Q. Let $V(S,A)$ be the vulnerability of the network to a particular attacker, A. Then Objective Function 3.2 shows the optimal network response given the current state of the attack.

The results of this proposed research will include a better understanding of how to respond to network security attacks. In addition the existing NIPAT

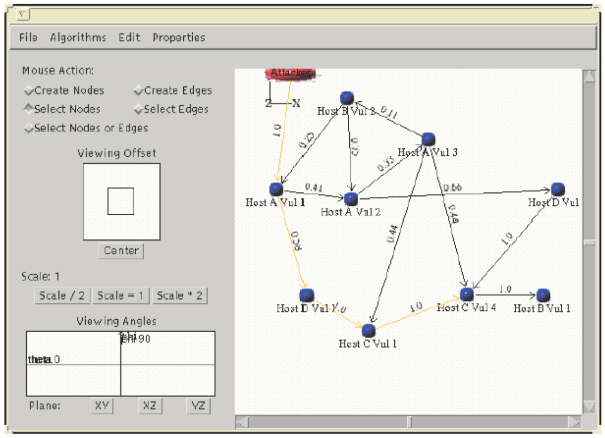

Figure 7. Various stages of attack.

will be incorporated with new algorithms, serving as experimental validation of the results from this project.

Failure of the grid-based approach

The grid-based approach, limited to the capabilities as previously discussed, has a considerable number of shortcomings. The first is the inability of the grid-based mechanisms, as presented above, to assign meaningful initial values that represent either security or insecurity. The current implementation of NIPAT uses scalar values that represent the "strength" of an attacker and the number of opportunities for an attacker to exploit a chain of *a priori* identified vulnerabilities. The reasoning in the development of NIPAT is that the strength of an attacker is a representation of the attacker's power in terms of combined instructions per second, advanced knowledge of the system under attack, and skill in the use of attack strategies. It has been proposed to augment NIPAT with vectors, where each element represents an attacker's strength in exploiting various predefined vulnerabilities. However, this assumes advanced knowledge of all possible vulnerabilities and the attacker's strength in exploiting each of those vulnerabilities. This is not a reasonable assumption for a system of even low complexity. Using a database, expert system, or object-oriented abstraction, to handle aggregations of vulnerabilities does not lead to a feasible solution because these mechanisms require that all possible vulnerabilities be known *a priori*. A more general vulnerability discovery and quantification technique is necessary.

It is our belief that many such tools, such as NIPAT, are salvageable as an information assurance design tools. The good qualities of NIPAT, such as safeguard optimization and likely attack path identification, particularly to lead an attacker to a fishbowl, are useful mechanisms for information assurance design. To provide a brief preview of our proposed solution for grid-based tools, consider the resistance in the electronic circuit analogy of information assurance as complexity where complexity and resistance are directly proportional. The relationship among vulnerability, resistance, and complexity is developed in more detail later in this report. In Section 3.3 we look at the properties required of a meaningful information assurance metric.

Fundamental properties and parameters of information

As discussed in the introduction, we desire to move the study of information assurance to a fundamental domain, where attacks need not be defined in advance. But what are the fundamental properties of information and how can we build upon them to achieve a science for the assurance of this information. We discuss below some basic properties of information that are candidates for fundamental parameters upon which to build.

***Size*—**In his ground breaking 1949 paper, Shannon introduces fundamental tradeoffs and limitations on the ability to transmit information across a channel disturbed by Additive White Gaussian noise (AWGN) [98]. This launched the science of information theory that has transformed the study of communications and coding of information, bringing the use of the term "bit" of information, which Shannon credits to J.W. Tuckey, into the mainstream literature. The idea that information can be quantized into bits (or sequences of yes or no answers to questions) is now well accepted, and one measure of the size of information is the number of bits used to convey the information. Information compression coding – both lossless and lossy– as well as forward error correction coding alter the size of the information in terms of bits by removing or adding redundancy. However, the unit of size, bits, is the term used to discuss the size of information, whether it is efficiently coded or not, error prone or self-correcting. Thus, while it is possible for information to change size without altering content, size is a fundamental property of information that should come into play under a set of fundamental laws of information assurance.

***Entropy*—**Shannon entropy [179] is a fundamental property of information that measures the uncertainty of a random variable X based on the probabilities of each outcome:

Entropy therefore relates to a source distribution of a random variable. Kolmogorov complexity is a related parameter that will be discussed in detail that relates to a specific sequence of information. These two parameters are extremely powerful properties of information that occur at the most fundamental level.

***Density, Mass or Energy*—**Density, mass and energy are properties of matter that have parallel and intuitively pleasing meanings in the domain of information. Density, like Kolmogorov Complexity, may measure the ability of a sequence to be compressed. Mass may simply represent the number of ones in a sequence, and energy as in thermodynamics may tie together quantities such as mass, density or entropy. The overriding goal is to find parameters that can be observed directly from the information sequences themselves and compare objective quantities on which to base the science of information assurance.

Towards complexity-based information assurance

Beginning with a high-level view of the problem definition, Figure 8 shows both secure manner of operation and insecure operation. Both manners of operation exist in the space of all possible forms of operation, M. Insecure operation, M_I, consists of those methods of operation that allow an information warfare aggressor entrance or access to control points into the information system. The intended secure operation areas M_S are well known, and some of the insecure paths are also known. Note that M_S and M_I can, and usually do, overlap. However, the entire area of operation can be extremely large and an exhaustive search for all insecure operation is not feasible. In Figure 8, Euclidean distance corresponds to the degree of security. This leads one to consider a metric space upon which to base information assurance. The initial approach assumes only that the metric has the characteristics of a metric in the mathematical sense as shown in Definition 3.1 where d is distance and p and q are points. Point p and point q have not been explicitly defined. As illustrated in the left side of Figure 8, an information system de-composed into many operating components could have a surface area as shown on the right side of Figure 8. Note that this surface is likely to change as a function of time; however, the time indices are not written for now. The points p and q are assumed to be relative to some absolute value; p and q can be security values in either different locations or at different time instances of the system. If d is a measure of security, then Definition 3.1 implies that there is no difference in security between the same point and itself; however, there must be a difference between any two distinct points in the security space. Definition 3.1 states that the measure between any two points in this space should be the same regardless of the order in which one takes the measurement. This means that, observed from a common vantage point, if security is measured at two different points in this space, p and q, then the measure of security will be the same regardless of the order in which the points are entered in the measure. It does not imply anything about the strength of an attack from p to q or an attack from q to p. It means, for example, that if p is less than q, then an attack from outside the system against p will be more likely to succeed than an attack against q. Finally, Definition 3.1 states that the distance between any two points will be less than or equal to the sum of the distances between each of those points and a common third point. Again, remember that this is a measure of security taken from a view outside the system of a potential attack from outside the system. As discussed in more detail in the remainder of this report, the actual measure will change as an attacker penetrates the system and as the attacker gains more knowledge of the system.

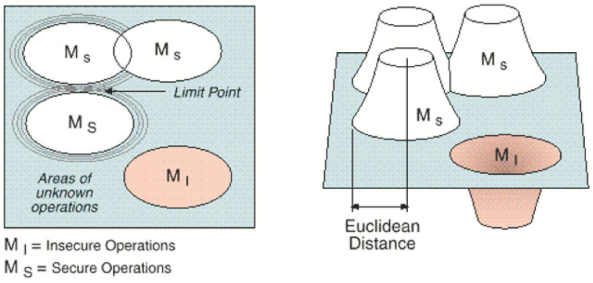

Figure 8. Set Theory View of Secure Operation.

In Figure 3 and Figure 4 a topographical and density plot shows the security of the system in Figure 2. These graphs are only suggested means of viewing information assurance, not a recommendation. Summing the attack strength at each node from all other nodes generates the graphs. Thus, the topology, or density, is the vulnerability of a particu-

lar area of the graph to all attacks. The light areas in the density plot and the higher areas in the topology map are areas of low vulnerability, while the darker areas or lower areas on the topology map are areas that are well secured. Remember that these are graphs of known vulnerabilities and the likelihood that they will be penetrated. The problem with these graphs is two-fold: what is the metric used to obtain the insecurity for each vulnerability, and how can it be assured that all vulnerabilities have been included in the graphs? Maps such as these require that Definition 3.1 must be satisfied.

Information assurance via set theory and complexity

If information assurance can be proven to reside in a metric space, or alternatively, if a metric space can be chosen in which information assurance can reside, then principles of mathematical analysis [100] can be used to rigorously determine more detailed characteristics. For example, M can be extremely large, possibly infinite. Are M_S, or conversely, M_F, open sets? If so, can limit points be defined? What does an open set mean with regards to information assurance and security? As a simple example, consider a password protection system. Each character that a legitimate user of the system adds to a password increases the number of possibilities that a brute force (non-dictionary) attack would require in order to guess the password. Thus, the longer the password, the more secure the system. While an infinite length password is not possible, the security does begin to approach a limit point. This can also be seen, for example, in any security safeguard that works via the addition of complexity (that is, adding more states to the Turing Machine to increase security). This approach towards safeguard design approaches a limit point but can never reach perfect security. However, in general, this appears to be the only known approach, and thus limit points must exist. Assurance is usually increased by increasing the apparent complexity of access to potential attackers while providing legitimate users the apparent, least complex, or in some sense, shortest, path to access of information. The complexity approach is carried forward in more detail in Section 6.

Topological space for information assurance

By definition, an open set, E, is one in which every point is an interior point. A point, p, is an interior point of E if there is a neighborhood, N, of p such that. A neighborhood $N_r(p)$ of point p consists of all points q such that where r is called the radius of the neighborhood. If security, as determined by a given metric, is an open set, then there are significant implications because of this. The best that can be hoped for in such a case is to determine limit points because a distinct boundary between security and insecurity would not exist. Will it be the case that adding layers of security is much like adding "open covers"; that is, the result can never be perfect security, but rather an approach to a limit point? The complement of an open set is closed; what does that imply for assessment of insecurity? NIPAT takes both probabilistic and maximum flow approaches to computing network insecurity flows. This tool is incomplete for at least two reasons: it assumes that all vulnerabilities have been identified and measured and that the vulnerabilities can be manipulated as discrete, closed sets. In order to determine whether such measurements can be applied to information assurance, consider topology, metric spaces, and the fundamentals of measurement theory in more detail. The definition below shows how the topology is induced by a metric d.

In the definition above is a collection of subsets of such that and, any finite intersection of members of is in, and any union of members of is in. For purposes of removing unnecessary detail, assume that the information system is a Turing Machine. Also assume that the vulnerability analysis tool in [43] displays vulnerabilities within the Turing Machine. Lemma 3.1 illustrates the topology that will be induced.

The intuitive notion is that d represents the ease of movement of an intruder from one vulnerability to another, where $d(x, y) : \mathrm{X} \times \mathrm{X} \rightarrow \Re$. A simple metric, as discussed previously, is to define d as the number of state / transition sequences within a Turing Machine representation of a system which an intruder can follow to move from vulnerability x to vulnerability y, or equivalently, the cardinality of the set of V from Definition 3.1. In this induced metric space, the NIPAT vulnerability tool can be considered an overlay of the Turing Machine representation of the system. The NIPAT tool filters out the state and transition details and shows only the direct connection among the vulnerabilities. Does information assurance reside within this metric space? One test would be whether the metric supports the design tradeoffs required in determining brittleness in the design of the system. To answer the above

question, let $A \subset V$ be the set of currently exploited vulnerabilities. Most information security approaches, including the one above, assume that all vulnerabilities have been discovered and measured. This can never be assumed to be the case. Performance, α, from Definition 6.1 is an open set, and as new security holes are discovered, $\alpha \frac{A}{\lim |A| \to \infty}$. If V represents vulnerability and is open, then secure operation, $\bar{V}$, is closed. Assume that $\sup_{x \in \bar{V}} d(x, x_0) < \infty$ for any x_0. Note that x is now an element of the set of secure operation. In other words, the number of secure operations is bounded. It is well known that a set is compact if and only if it is closed and bounded. Next, Section 4 takes a closer look at a model for Information Assurance upon which our new metric is based.

2.3 AN INFORMATION ASSURANCE MODEL

In order to develop a reference and working model for our exploration into the fundamentals of cybersecurity and cyber-physics, a Turing Machine [110]is used to characterize system operation. The Turing Machine is one of the most fundamental general computing abstractions and is well-known in computer science. It has a rich theory of its own that this report intends to utilize to its advantage. The Turing Machine consists of a seven-tuple (Q, T, I, δ, b, q_0, q_f). Q is a set of states, T is a set of tape symbols, I is a set of input symbols, b is a blank, q_0 is the initial state, q_f is the final state. δ is the next move function. δ maps a subset of $Q \times T^k$ *to* $Q \times (T \times \{L, R, S\})^k$. L,R, and S indicate movement of the tape to the left, right, or stationary respectively. There can be multiple tapes. Thus δ implements a "next move" function. Given a current state and tape symbol, δ specifies the next state, the new symbol to be written on the tape, and the direction to move the tape. The sets of symbols that lead to an accepting state (q_f) is the input language ($\sum$). One approach to the study of security is to consider the Turing Machine representing normal operation of an information system. In such an approach, if the Turing Machine recognizes, or accepts, an input language, then a user has gained access to the system. If the Turing Machine accepts a language that we did not anticipate ($\sum_I$), then the system is vulnerable, as stated in Hypothesis 4.1.

Clearly, the Turing Machine is an abstract representation of any protocol implementation, or operating component operation. The set of unanticipated input languages that is accepted is the vulnerability of the component, V, as shown in Definition 4.1. This is illustrated in an existing tool GE Research has developed [43]which displays system vulnerabilities from a Unix operating system running Internet Protocol data communications. This work requires a definition of security, shown in Definition 4.1. Our use of a metric space requires us to prove Definition 3.1 holds.

In order to make the problem of quantifying assurance tractable, consider the fundamental assurance characteristics of an individual Turing Machine. Assume all internal operations are perfectly secure. This follows from [110] in which Turing notes that machine operations are atomic.[1] It is assumed also that reading, writing, tape control, and state control cannot be observed, modified, or interfered with in any manner. The only effect upon the system is the input language. A malicious input language, given a single Turing Machine, can cause denial of service simply by never halting. Also it is assumed that the output tape is accessible to the world. Thus a malicious input language could write secret information to the output tape. What happens with multiple users each with their own regulated data access?

Another objective of an attack may be to determine the function performed by the Turing Machine. In this case the attacker is assumed to have the ability to enter every member of the input language in order to deduce operation by viewing the output. The attacker is actually deducing δ. Deducing process versus data is described from a complexity viewpoint in more detail in Section 6.3. A Trojan horse is implemented either by allowing an attacker access to modify another user's input tape, or perhaps δ of the machine itself.

Given a network of Turing Machines, in which one machine's input is the output of another machine, an input language could be self-replicating, that is, a virus. However, Turing Machines are composable. A single machine with additional states can implement any set of Turing Machines. This implies that a state-based approach should be taken in the analysis of assurance. For example, is state q_x more or less secure that state q_y? How can this com-

1. In [110], Turing casually notes the symbols on the machine's tape can form a conditionally compact space.

parison be made within a single Turing Machine? Is the state on a path that leads to an insecure event? An individual state by itself does not reveal much about the current assurance level. However, a set of states, considered along a continuum, provides much more information. Can a gradient be established that leads to the likelihood of an insecure event? Such a gradient requires a well-defined metric, which is what this work is leading towards. One approach towards computing vulnerability would require checking every possible input language in order to quantify relative insecurity levels. This is obviously an intractable approach. This problem is tackled by means of complexity in Section 6.

In order to begin to understand how information assurance can be quantified, consider the manner in which systems that implement information assurance can be designed. Design involves the tradeoff of one benefit for another. Brittle Systems provide a framework for understanding the tradeoffs in performance versus failure of information systems. Brittle systems analysis [108] is based on the idea that systems can fail in a manner analogous to brittle fracture. A system can maintain very high performance until it fails quickly and catastrophically, as illustrated by performance curve P_h Figure 9 or sys-

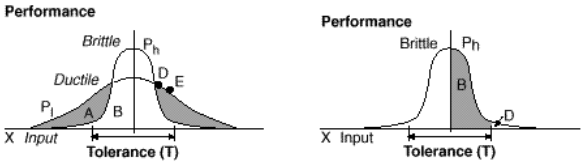

Figure 9. Definition of Brittleness.

tems may fail by exhibiting lower performance in a gradual, more ductile manner as in curve P_l. The mapping between Brittle Systems Theory and information assurance is shown in Table 2. This analysis can be directly applied to the Turing Machine. Changes in any of the state machine parameters, Q, T, I, δ, b, q_0, q_f, may modify the brittleness of the system. For example, addition of a new state and transition could cause the system to behave in a more ductile or brittle manner. What is the measure of performance in a Turing Machine model of information assurance? What does catastrophic failure mean in a Turing Machine model of information assurance? Performance is α from Definition 5.2 and X is measured related to attacker effort. The answers to the above questions are intimately linked to the choice of metric. Based on Definition 3.1 discussed previously, one could choose the metric to be the number of state / transition paths available to an attacker to reach a particular target state, or equivalently, the cardinality of the set of languages, V, given in Definition 3.1. Another possible metric could be the proximity of the attacker's current state to the target state. Various other complex metrics could be contrived such as ones that include the work involved for an attacker to move from one state to another. These complexity metrics are based upon a known attacker at a given state. If a single measurement is required to describe the performance of the information assurance system, then values generated by the choice of metric must be combined in a reasonable manner. However, a single value is not meaningful in the same way that capacity would be useful in determining load, unless there were a meaningful attacker strength with which to operate. Next, in Section 5, more detail on Brittle Systems and how they relate to a new information assurance metric are discussed.

2.4 BRITTLE SYSTEMS, DETERMINISTIC FINITE AUTOMATA, AND VULNERABILITIES

A Deterministic Finite Automaton (DFA) consists of a 5-tuple (S, I, δ, s_0, F) where S is the set of states, I is the input alphabet, δ is a mapping from into I, s_0 is the start state, and F is a subset of S called the final, or accepting states. A DFA is less powerful than a Turing Machine in terms of the languages it can recognize as well as less capability in performance of general computation. However, DFA have been well studied and facilitates a framework in which new theories related to Information Assurance can be studied. An example of Brittle Systems using Definition 3.1 for vulnerability is illustrated for the DFA shown in Figure 10. A single vulnerability is represented as a single modified transition. The modified transition represents an error in either the design or implementation that allows an attacker to penetrate the system. The effect of each transition modified from its original source node to each possible destination node in the automaton is exhaustively checked. The effort expended by an attacker is assumed to be proportional to the length of the strings used in the language. $P = \alpha$ (Definition 5.2) and X is the effort of an attacker measured in terms

Table 2 Brittle System Definitions

Materials	Science Brittle Systems	Information Assurance
Stress	Amount parameter exceeds its	tolerance
Applied force under the weight of an	attack	Toughness
System robustness	Encryption strength and sensitivity of intrusion detectors	Ductility
Level of Performance outside	Tolerance	Ability of system to gracefully degrade given an attack
Plastic Strain	Degradation from which the	system cannot recover
Trojan horse	Brittle Fracture	Sudden steep decline in
performance	Sudden catastrophic collapse of all information assurance	Young's Modulus
Amount tolerance exceeded over degradation		Deformation
Degradation in performance	The amount by which vulnerability has	been increased due to an attack
Brittleness	Ratio of hardness to ductility	
Ductile Fracture	Graceful degradation in performance	Ability of information to gracefully degrade under an attack
Reversible Strain	Degradation from which the system can recover	Trojan horse detection and removal
Hardness	Level of performance within tolerance limits	Resistance to decryption

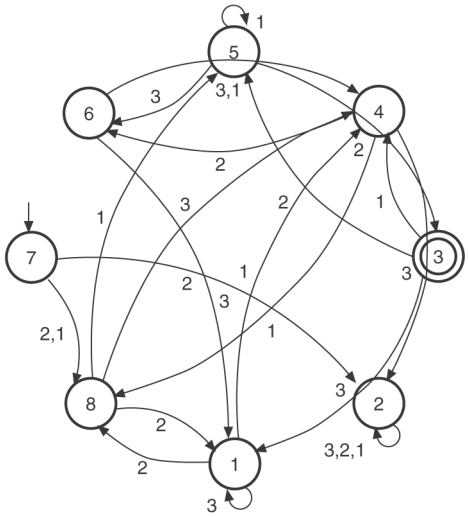

Figure 10. Example of Deterministic Finite Automation.

of language size required to reach an unintended accepting state. The algorithm requires starting with the actual system as represented in Figure 10, modifying a transition and then recording the number of additional strings accepted. This is repeated for each transition in the base system. As shown in Figure 11,

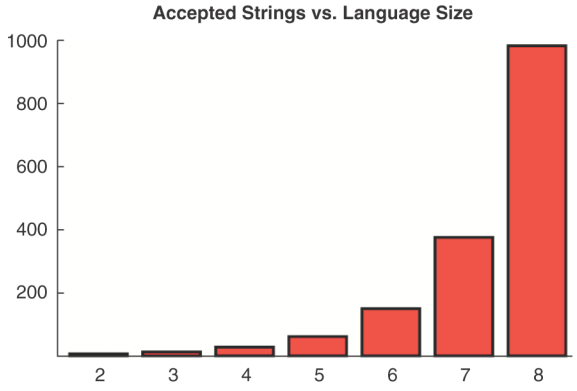

Figure 11. Ductile Vulnerability.

a modification of the transition from State 7, Input 3, Destination State 2, (7,3,2) yields a small number of vulnerabilities at string length two with a

maximum of 1000 vulnerabilities at string length 8. This performance is ductile compared to the graph shown in Figure 12, where transition (1,3,1) is modified. Figure 12 shows more brittle behavior because it takes a longer string length, thus more effort by the attacker to find vulnerabilities; however, the vulnerability increases rapidly as the string length increases. A more precise definition of Brittleness is given in Definition 5.1. The discrete form of brittleness is accomplished by normalizing the sum of the number of strings accepted of systems A and B to be the same, then summing the value where B exceeds A as in the left side of Figure 13.

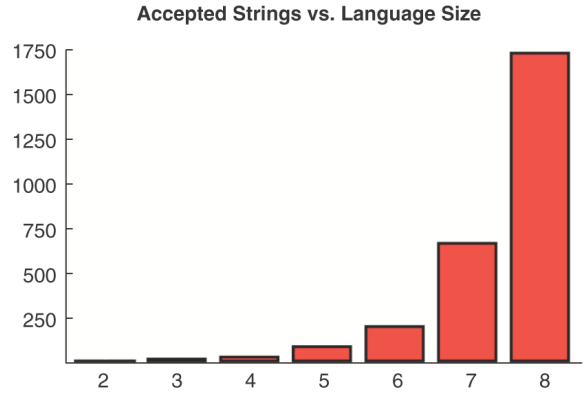

Figure 12. Brittle Vulnerability.

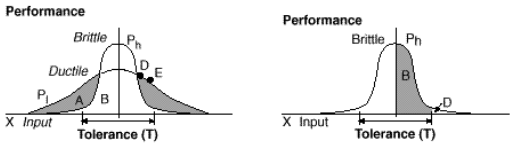

Figure 13. Definition of Brittleness.

Building upon Definitions 3.1 and 3.2 requires that Turing Machine states, Q, be identified as either secure or insecure. If an attacker can reach a member of $q_{insecure}$ then the attacker is considered to have performed a successful attack. If an attacker can never reach a member of $q_{insecure}$ then the system is considered invulnerable. The challenge is that neither the attacker nor the defender knows the entire structure of the Turing Machine's program because the attacker is unlikely to have complete knowledge of the defender's system and because even the defender may not fully understand the system that was developed. The unknown behavior of a system is discussed later in terms of Apparent Complexity in Section 6.5.

Definition 5.3 provides a means for easily computing complexity in the world of finite automata. Next the relationship between brittleness and complexity is addressed. One might intuit that a faulty transition in a less complex automaton will have less of an impact than a faulty transition in a complex version of the equivalent automaton. The definition of equivalent automata is given in Definition 5.4.

The simplicity, intended to be the opposite of complexity, is given in Definition 5.5 as the difference in size between the current implementation of an automaton and its minimized size.

A simple implementation of an automaton has more transitions and states than necessary to implement the automaton. Thus, there is more opportunity for an attacker to find a weak point in the system. However, once an attacker breaks into a simple system, there will be, on average, more energy, that is, longer string length, required to reach the attacker's destination. Thus, greater simplicity should imply reduced brittleness (Hypothesis 5.1).

Figure 14 and Figure 15 show a simple and complex implementation, respectively, of the same arbitrary information system. Figure 14, as a simple implementation, is what might be intuitively referred to as an inefficient implementation, with many more states than necessary. This yields the opportunity for more vulnerabilities and faults. However, it also takes the attacker more effort to

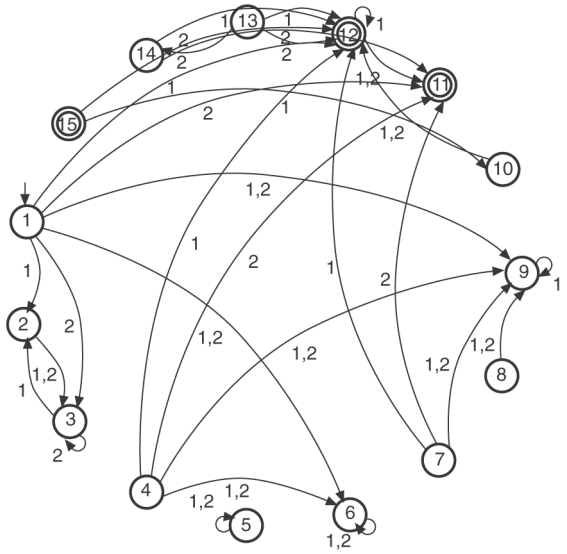

Figure 14. A Simple DFA.

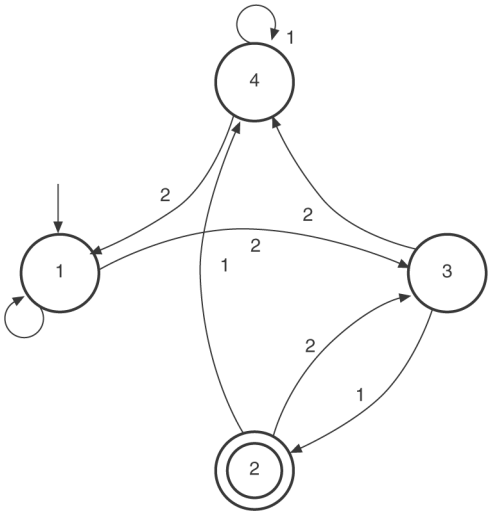

Figure 15. Complex Version of the DFA Shown in Figure 14.

reach a given target. Figure 15 is a closer representation of the true complexity of the same system. It has fewer opportunities for failure; however, the failures that occur will be more significant.

In Figure 16 and Figure 17, brittleness and complexity are compared. Brittleness is computed as defined in Definition 5.1. Performance is defined based upon the number of accepted strings and language size. The ratio of the number of accepted strings to total language size is inversely proportional to the performance. For each possible fault, this ratio is compared to a consistent base case consisting of an exponentially growing number of accepted words as the language size increases. A brittle system accepts few words initially, then suddenly accepts a large number, while a ductile system accepts a moderate, but gradually increasing number with no sudden increase. The brittle measure is graphed as a function of a fault in the state specified on the dependent axis. A fault is the disappearance of a state that results in the direct connection of a transition to the destination nodes of the faulty node. Complexity is estimated as the number of transitions in the smallest representation of the resulting faulty system. Comparing Figure 16 and Figure 17, there appears to be an opposite relationship between brittleness and complexity. That is, a system with greater complexity results in lower brittleness. Greater complexity indicates a larger number of transitions and states exist, thus there is more opportunity for an attack, but more effort is required by the attacker to successfully complete the attack. In Figure 18 and Figure 19 a similar analysis is performed on the more compact, or truer representation of the complexity, of the same system. Notice that the system with an implementation that is closer to its true complexity is much more brittle. Also, note that the inverse relationship between

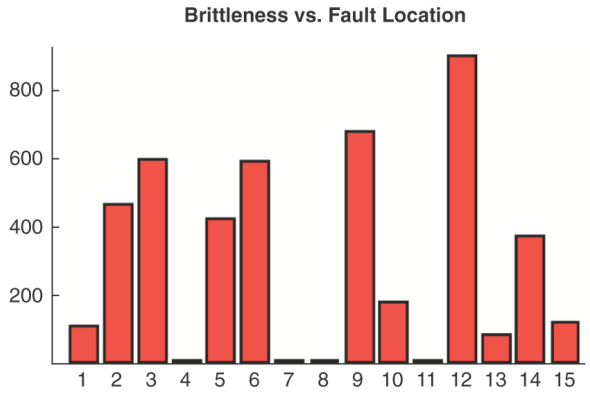

Figure 16. Brittle Measure of DFA Shown in Figure 14.

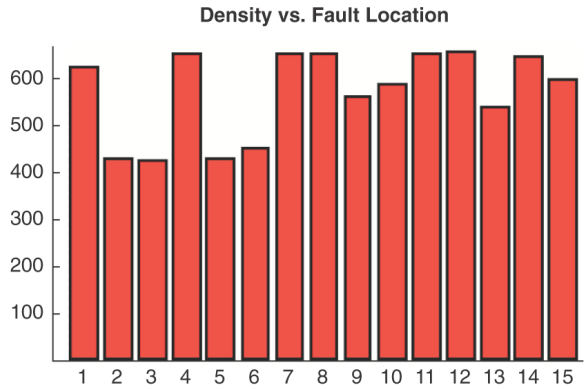

Figure 17. Complexity of DFA Shown in Figure 14.

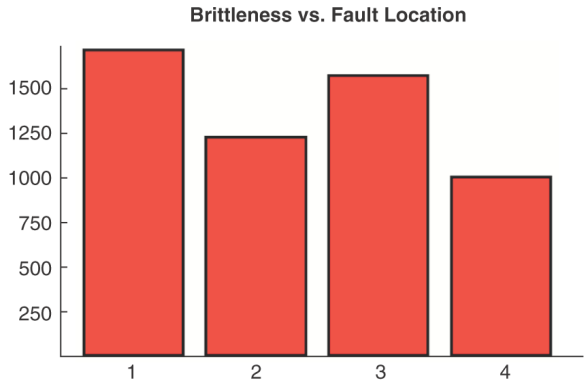

Figure 18. Brittle Measure of System Shown in Figure 14.

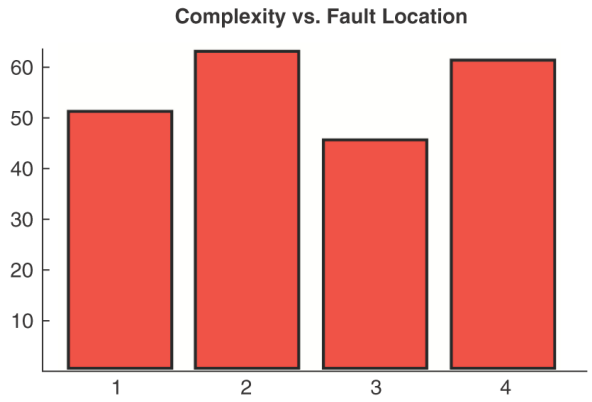

Figure 19. Complexity Measure of System Shown in Figure 14.

complexity and brittleness holds in the more complex system as well.

An important result in this exploration of the relationship among vulnerability, complexity, and brittleness is that the larger the system, in terms of the number of transitions and states, the lower its brittleness. This suggests that larger systems, requiring traversal of larger numbers of states and transitions to reach an accepting state, or attack target, require more effort to successfully attack. A system that has a large amount of inherent complexity cannot be made any more compact than its Kolmogorov Complexity, which is discussed later. An intelligent attacker may be able to observe an inefficiently implemented system and reduce it to its most compact form, that is, its Kolmogorov Complexity, thus easily identifying paths of attack to reach specific targets. A truly safe system is thus obtained, not by building inefficiency in the system, but rather, by making the view to the attacker as inherently complex as possible.

2.5 KOLMOGOROV COMPLEXITY

Information must be accessible to legitimate users while access is denied to potential attackers. This is done by increasing the apparent complexity of access to information and by providing legitimate users with enough *a priori* knowledge to reduce the apparent complexity. This leads one to conclude that complexity itself would be a useful metric for Information Assurance. However, the search for an absolute measure of complexity is a problem that may be equally as hard as quantifying Information Assurance itself. There is a good reason for this; they are, in a sense, one and the same thing. To show this, begin with the definition of complexity. Kolmogorov complexity is a measure of descriptive complexity that refers to the minimum length of a program such that a universal computer can generate a specific sequence. Kolmogorov complexity is described in Definition 6.1, where φ represents a universal computer, p represents a program, and x represents a string. Universal computers can be equated through programs of constant length; thus a mapping can be made between universal computers of different types.

Kolmogorov Complexity is proposed as a fundamental property of information that has properties of conservation that may be exploited to provide information assurance. In this report, Kolmogorov Complexity is reviewed and current work in this area explored for possible applications in providing information assurance. The concept of Minimum Message Length is explored and applied to information assurance, yielding examples of possible benefits for system optimization as well as security that can be achieved through the use of Kolmogorov Complexity based ideas. Finally, complexity based vulnerability analysis is demonstrated through simulation in Section 9.

Currently information security is achieved through the use of multiple techniques to prevent unauthorized use. Encryption, authentication / password protection, and policies all provide some level of security against unauthorized use. But other than simply relying on these secure barriers, how does one measure the health of a security system. If a password or encryption key is compromised, what indication will be available? The degree to which a system is compromised is difficult to ascertain. For example, if one password has been guessed, or two encryption keys determined, how secure is the information system? Are all detectable security issues equal, or are some more important than others? These difficulties reflect the fact that there is no objective, fundamental set of parameters that can be evaluated to determine if security is maintained. Insecurity may not be detected until an absurd result (rat in a tank) discloses the presence of an attacker. An inherent property of information itself is desired that can be monitored to ensure the security of an information system. The descriptive complexity of the information itself – the Kolmogorov Complexity – is a strong candidate for this purpose.

Complexity and vulnerability in information assurance

Progress in information assurance cannot proceed without fundamental measures. Measurement requires that information assurance be identified and quantified. In order to make progress towards this goal the results of a study in the evolution of the complexity of information are presented. An underlying definition of information security is hypothesized based upon attacker and defender as reasoning entities, capable of innovation. This leads to a study of the evolution of complexity in an information system and the effects of the environment upon the evolution of information complexity. Understanding the evolution of complexity in a system enables a better understanding of where to measure and how to quantify vulnerability and should lead towards a calculus of information system complexity. Finally, the design of the tool under construction for automated measurement of information assurance, used to gather and analyze the complexity data in this report, is presented. The motivation for complexity-based vulnerability analysis comes from the fact that complexity is a fundamental property of information. If the interaction of information complexity with its environment can be understood, then a new understanding of information assurance may be possible, one in which assurance can be better understood and measured. Quantification is necessary because tools have been developed to measure and analyze security assuming that rigorously defined security metrics exist. Unfortunately, such metrics do not yet exist.

One method for examining information assurance is to consider its converse, insecurity and vulnerability. Vulnerability analysis tools today require types of vulnerabilities to be known *a priori*. This is unacceptable, but understandable given the challenge of finding all potential vulnerabilities in a system. Information assurance is a hard problem in part because it involves the application of the scientific method by a defender to determine a means of evaluating and thwarting the scientific method applied by an attacker. This self-reference of scientific methods would seem to imply a non-halting cycle of hypothesis and experimental validation being applied by both offensive and defensive entities. Information assurance depends upon the ability to discover the relationships governing this cycle and then quantify and measure the progress made by both an attacker and defender. This work attempts to lay the foundation for quantifying information assurance in such an environment of escalating knowledge and innovation.

Any vulnerability analysis technique for information assurance must account for the innovation of an attacker. Such a metric was suggested about 700 years ago by William of Occam [94]. Occam's Razor has been the basis of much of this invention and the complexity-based vulnerability method to be presented. The salient point of Occam's Razor and complexity-based vulnerability analysis is that the better one understands a phenomenon, the more concisely the phenomenon can be described. This is the essence of the goal of science: to develop theories that require a minimal amount of information. Ideally, all the knowledge required to describe a phenomenon can be algorithmically contained in formulae, and formulae that are larger than necessary indicate lack of a full understanding of a phenomenon. The working hypothesis in this report is that vulnerabilities are locations of low complexity.

Next consider the attacker as a scientist trying to learn more about his environment. In this case, the environment is an Information System. The attacker as scientist will generate hypotheses and theorems. Theorems are attempts to increase understanding of the universe by assigning a cause to an event, rather than assuming all events are random. From [94]and Definition 5.1 above, if x is of length $l(x)$, then a theorem of length $l(m)$, where $l(m)$ is much less than $l(x)$, is not only much more compact, but also $2l(x)-l(m)$ times more likely to be the actual cause than pure chance. Thus, the lower the complexity of the theorem, as stated by Occam's Razor, the more likely the theorem is to be correct.

Consider Figure 1 and Figure 2and notice how one intuitively views current as an attacker's strength and voltage differential as the desire or pressure of an attack upon a system. Resistance is intuitively viewed as the ability of the system to block an attack. Thus, following this intuition, one can view vulnerability as inversely proportional to resistance and directly proportional to the complexity of the system as viewed by an attacker. Capacitance and inductance might be intuitively viewed as relating to the brittleness of the system. Brittle Systems analysis is discussed later; however, it relates to the tradeoff in performance and degradation of a system. This intuition is captured in Hypothesis 6.1.

This hypothesis is significant because it sets the direction of our efforts and determines the fundamental basis upon which the remaining work rests. As the attacker is refining the theorems (that is, reducing their complexity), the defender is attempting to raise their complexity, while still maintaining (low complexity) access for legitimate users. Thus there is an ever-increasing cycle of complexity (see Figure 20). It is as though while a scientist studies

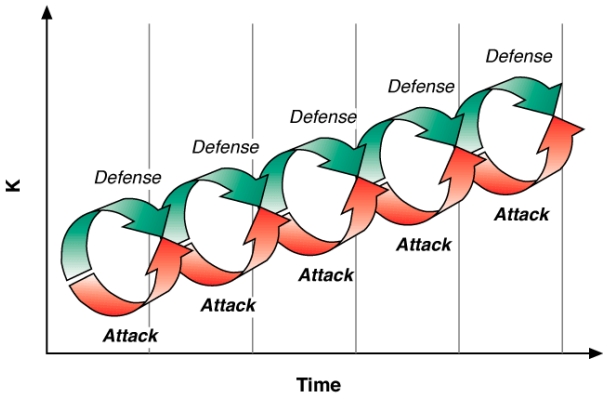

Figure 20. Evolution of Complexity Caused by Attack and Defense.

natural phenomenon, nature actively tries to hide its true internal operation through the addition of more complexity.

Kolmogorov Complexity is a measure of descriptive complexity contained in an object. It refers to the minimum length of a program such that a universal computer can generate a specific sequence. A good introduction to Kolmogorov Complexity is contained in [118] with a solid treatment in [97]. Kolmogorov Complexity is related to Shannon entropy, in that the expected value of $K(x)$ for a random sequence is approximately the entropy of the source distribution for the process generating the sequence [118]. However, Kolmogorov Complexity differs from entropy in that it relates to the specific string being considered rather than the source distribution. Kolmogorov Complexity can be described as follows:

Random strings have rather high Kolmogorov Complexity – on the order of their length – as patterns cannot be discerned to reduce the size of a program generating such a string. On the other hand, strings with a large amount of structure have fairly low complexity. Universal computers can be equated through programs of constant length; thus a mapping can be made between universal computers of different types, and the Kolmogorov Complexity of a given string on two computers differs by known or determinable constants. The Kolmogorov Complexity $K(y|x)$ of a string y given string x as input is described by Definition 6.2: where $l(p)$ represents program length p and φ is a particular universal computer under consideration. Thus, knowledge or input of a string x may reduce the complexity or program size necessary to produce a new string y.

The major difficulty with Kolmogorov Complexity is that it cannot be computed. Any program that produces a given string is an upper bound on the Kolmogorov Complexity for this string, but the lower bound [97] cannot be computed. A best estimate of Kolmogorov Complexity may be useful in determining and providing Information Assurance due to links between Kolmogorov Complexity and information security that will be discussed later. Various estimates have been considered, including compressibility, or pseudo-randomness, which measure the degree to which strings have patterns or structure. A new metric that is related to the power spectral density of the sequence auto-correlation is introduced in a later section. However, all metrics are at best crude estimates. The inability to compute Kolmogorov Complexity persists as the major impediment to widespread utilization.

Despite the problems with measurement, Kolmogorov Complexity and information assurance are related in many ways. Cryptography, for example attempts to take strings that have structure and make them appear randomly. The quality of a cryptographic system is related to the system's ability to raise the **apparent complexity** of the string, an idea discussed in detail later, while keeping the actual complexity of the string relatively the same (within the bounds of the encryption algorithm). In other words, cryptography achieves its purpose by making a string appear to have a high Kolmogorov Complexity through the use of a difficult or impossible to guess algorithm or key. Security vulnerabilities may also be analyzed from the viewpoint of Kolmogorov Complexity. One can even relate insecurity fundamentally to the incomputability of Kolmogorov Complexity and show why security vulnerabilities exist in a network. Vulnerabilities can be thought of as the identification of methods to accomplish tasks on an information system that are easier than intended by the system designer. Essentially the designer intends for something to be hard for an unauthorized user and the attacker identifies an easier way of accomplishing this task. Measuring and

keeping track of a metric for Kolmogorov Complexity in an information system provides a method to detect such short-circuiting of the intended process. Note that the definition of complexity rests upon the notion of the Turing Machine, which is also the basis of our information assurance model. One direction in this research is to combine the definition of complexity with the definition of vulnerability in a fundamental manner. This has been done using Brittle Systems and Hypothesis 6.1. Next, Section 6.2 examines estimates of Kolmogorov Complexity.

Measures of information complexity

This section looks at proposed estimates of Kolmogorov Complexity. The microscopic approach to the study of information complexity evolution begins by considering the change in complexity of a single interaction. Later this report expands the results to a larger scale and discusses the results. However, in order to make the problem of estimating complexity tractable, two approaches are used. The first approach is based upon Finite Automata. A Finite Automata whose smallest accepted string is the bit-string whose complexity is to be determined, shown in Definition 6.3, is minimized using techniques such as [79,80] where *L(FA)* is the set of languages accepted by the automaton *FA* and *l(FA)* is its size.

Figure 21 illustrates the uncompressed representation of an arbitrary bit-string, 101_2. This automaton accepts many other bit-strings in addition to 101_2; however, the size of the minimized automaton that accepts that bit-string, based upon the number of transitions, for example, is an estimate of the complexity of 101_2. Thus, minimizing 101_2 also minimizes other bit-strings such as those formed by the regular expression $0^*1^+0^+1_2$ where the * indicates zero or more of the preceding symbol and + indicates one or more of the preceding symbol. However, 101_2 is the smallest string accepted by the automaton.

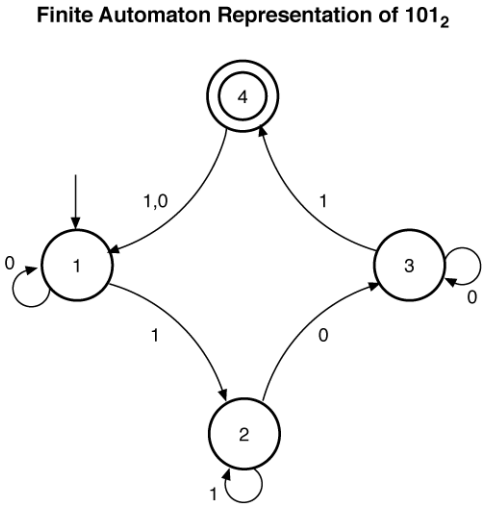

Figure 21. Finite Automation Representation of 101_2.

The Mathematica function *UnionAutomata* returns an Automaton that is the union of two automata. Notice that the complexity of the union of two automata is less than the sum of the complexity of each automaton as shown in Figure 22,

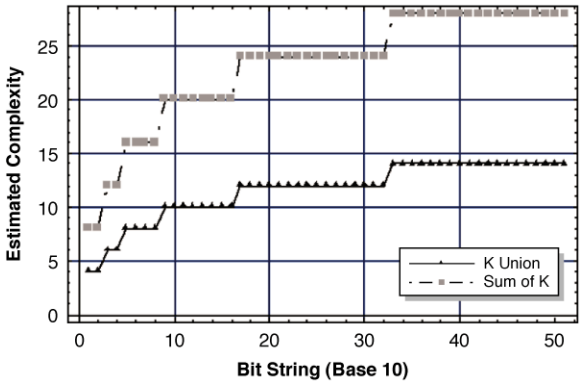

Figure 22. Complexity of Union of Automata versus Sum of Complexities.

graphed as a function of bit-strings representing the base ten integers from 1 to 50. This validates a known theorem from complexity theory discussed next, namely, that the resulting bit-string complexity from a program is less than or equal to the sum of the complexity of the input bit-strings, the length of the program, and a constant. The complexity of the combined bit-strings should be less than the sum of the automata complexities plus a constant that is dependent upon the size of the Universal Turing Machine, sometimes expressed as $H(X\,Y) \leq H(X) + H(Y) + c$ where $H()$ is the size of the smallest program capable of computing a specified result. Another way in which to view individual or microscopic interaction is to consider that if a program p of length $L(p)$ takes input string x to produce output string y, that is, $y = p(x)$, then Definition 6.4 defines the microscopic change in complexity where K is the Kolmogorov Complexity and c depends upon the underlying Universal Turing Machine [137].

Another approach for estimating complexity used in this report is based upon compression. The

inverse compression ratio, the ratio of the compressed size to the original length, is used as an estimate of the complexity. A highly complex bit-string cannot be compressed as much as a low complexity bit-string. A plot of automata-based complexity versus compression-based complexity is shown in Figure 23. The compression formulae used is where

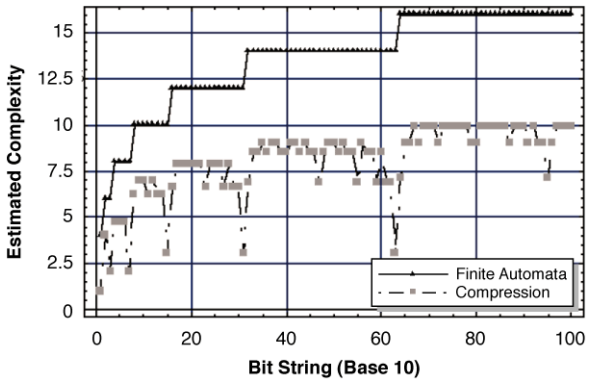

Figure 23. Automata and Compression-Based Measures of Complexity.

$H()$ is the entropy, n is the bit string length, w is the number of one bits, and c is the size in bits needed represent the length of the bit-string. Clearly, the compression-based mechanism provides a more accurate measure of complexity in the Kolmogorov sense; however, the automata-based mechanism has advantages in that automata provide a simplified and convenient mechanism for reasoning about computation at the microscopic level.

As previously discussed, due to its non-computable nature, estimates of $K(x)$ are difficult. Numerous techniques for estimating $K(x)$ are discussed in [97]. The task of estimating K is related to the task of assessing string structure. We now introduce a new primitive approach to this related issue based on the power spectral density of a string's auto-correlation. This approach highlights the ability to gain knowledge of $K(x)$ without any higher knowledge about the system producing string x or the meaning of the information.

Recognizing that the complexity of a binary string may be defined in many ways. A useful complexity measure may be related to properties of the string's non-cyclic auto-correlation. Specifically, given an n-bit binary string, S, where

$$S = \{s(i)\}, 0 \le i < n \tag{1}$$

and

$$s(i) \in \{\pm 1\} \forall i \tag{2}$$

define the non-cyclic auto-correlation, R, as

$$R = \{r(i)\}, 0 \le i < n \tag{3}$$

where

$$r(i) = \sum_{j=0}^{n-i-1} s(j)s(i+j) \tag{4}$$

From R, calculate the sequence's non-negative power spectral density, Φ_i, by multiplying the Fourier transform of R by its conjugate. The measure for binary string complexity that is formed is denoted by Ψ and is defined as

$$\Psi = \frac{1}{\text{nrm. factor}} \sum_i \Phi_i \log \Phi_i \tag{5}$$

The motivation to this approach is found in the rich and venerable field of synchronization sequence design. Sequences that have an auto-correlation whose side-lobes are of very low magnitude provide good defense against ambiguity in time localization. Such an auto-correlation function will approximate a "thumbtack" and its Fourier transform will approximate that of band-limited white noise.

The authors of this report expect that Ψ will be of utility in assessing complexity as it relates to the compressibility of a binary string. To begin the testing of this hypothesis, we generated strings from the Markov process diagrammed in Figure 24.

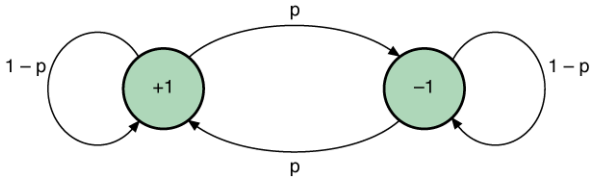

Figure 24. Markov Model for String Generation.

A series of binary sequences of 8000 bits was generated; each for different values of p. Ψ was computed for each of these strings and also packed into 1000-kilobyte files. These were subjected to the UNIX compress routine. The Inverse Compression Ratio (ICR) was computed which is the size of the compressed file normalized to its uncompressed size, 1000 kilobytes in these cases. The hypothesis is that Ψ and the ICR should vary in a similar manner and that Ψ might be a useful measure of sequence compressibility and hence complexity. The graph in Figure 25 following seems to endorse this hypothesis and further research is motivated.

The above results show that fundamental parameters such as power spectral density of sequence auto-

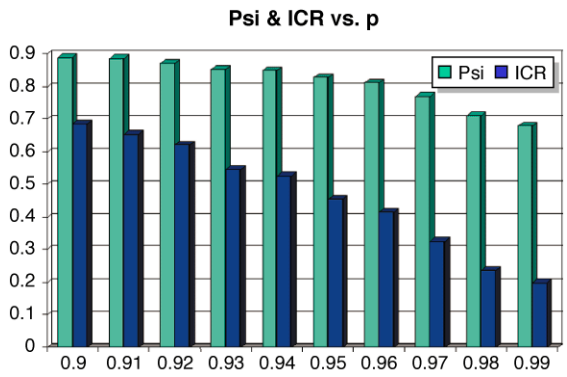

Figure 25. Variation of Psi and ICR with *p*.

cor relation and compressibility are related and follow similar trends. These fundamental metrics are possible candidates for measuring trend of increase or decrease in $K(x)$. However, also illustrated by these results (the unequal rate of change between the two metrics) are the loose bounds within which estimates of $K(x)$ are related. Other methods of estimating $K(x)$ are described in [97]. In the next section we introduce a method for attacking the issue of loose bounds in order to make complexity metrics useful for the purposes of assessing and providing information assurance.

Since it is not computable, few applications exist for Kolmogorov Complexity. One growing application is a statistical technique with strong links to information theory known as Minimum Message Length (MML) coding [143]. MML coding encodes information as a hypothesis that identifies the presumptive distribution, from which data originated, appended with a string of data, coded in an optimal way. The length of an MML message is determined as follows: $\#M = \#H + \#D$, where $\#M$ is the message length, #H is the length of the specification of the hypothesis regarding the data, and $\#D$ is the length of the data, encoded in an optimal manner given hypothesis H. As discussed in [143], MML coding approaches the Kolmogorov Complexity or actual bound on the minimum length required for representing a string of data.

Process vs. data complexity

Complexity can be applied to the problem of information assurance in two ways. As discussed above, conservation of apparent complexity may enable detecting and correcting abnormal behavior. Another method of using apparent complexity for information assurance is in the identification of weak areas or vulnerabilities in the system. Consider the postulate that the more apparently complex the data, the more difficult for an attacker to understand the data and exploit the system. Thus, the more apparently complex, the less vulnerable it is and vice versa. One proposed metric for vulnerability relates to evaluating the apparent complexity of the concatenated input and output $K(X.Y)$. This relates to the joint complexity of the data input and output from a certain process (Black Box). The lower the complexity of $K(X.Y)$ the easier the data is for an attacker to understand; thus we will regard $K(X.Y)$ as a measure of data vulnerability. A competing metric is the relative complexity $K(Y|X)$ of the process. This is the "work" done on the information by the process, or the complexity added or removed from X to produce Y. Thus, $K(Y|X)$ is a measure of process vulnerability. The relationship among this set of complexity metrics and the black box process is shown in Figure 26.

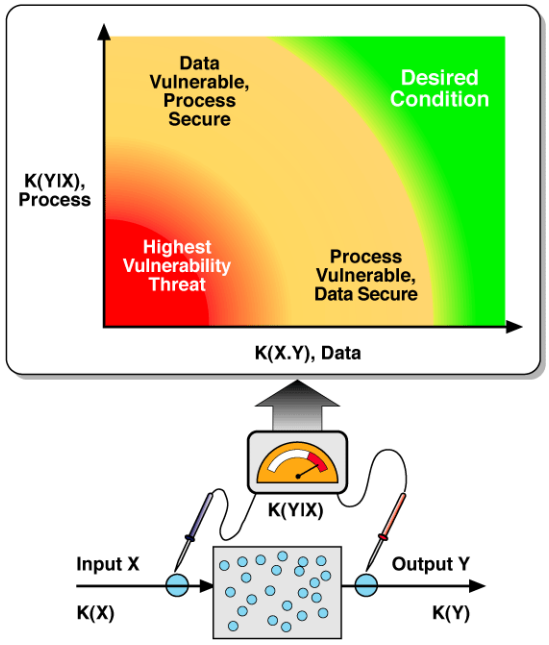

Figure 26. Process versus Data Vulnerabilities.

Data vulnerability relates to how vulnerable a system is to an attacker knowing information. This type is perhaps best measured by $K(X.Y)$, where the cumulative complexity of input and output data is observed to measure the difficulty an attacker would face in decrypting or identifying messages contained

in input and output. For example, hopefully K(encrypted message) appears much greater than K(decrypted message) to the casual observer and is only recognized to be on the order of K(decrypted message) to an authorized user with the correct key after the decryption algorithm has been run.

Process vulnerability relates a system's susceptibility to an attacker understanding the processes that manipulate information. This vulnerability is best quantified by the complexity injected or removed from the data by the process at work. For example, a copy or pass through process adds little complexity, $K(Y|X)$ is zero. But if encrypted data is sent through the copy process, $K(X.Y)$ will be high. The attacker will be unable to discern the messages that are sent, but can learn to perhaps simulate this particular black box quite effectively. Whereas if plain text data is sent through the copy process $K(X.Y)$ will be low, and in addition to understanding the process at work, and attacker may be able to know the particular messages that are sent. Both vulnerabilities are undesirable and represent two different dimensions of vulnerability to be avoided. To make systems secure one must maximize both process and data complexity to a non-authorized user while keeping the systems simple to authorized users. Proper accounting of $K(Y|X)$ and $K(X.Y)$ throughout the system will enable both identification of weak areas as well as identification of foul play through the conservation principles discussed earlier.

Vulnerability reduction by means of system optimization

In this section we discuss issues related to system optimization that can be achieved through Kolmogorov complexity and various tradeoffs. Compression and security are strongly linked in that they are bounded optimally by the most random sequence that can be produced. But smallest program size is not the only or even most important performance metric. Execution time must also be considered. The tradeoff is indicated in Figure 27.

Through the use of active network techniques [3] the tradeoff indicated may be dynamically addressed using a concept called Active Packet Morphing for network optimization. As shown in Figure 28, by changing the form of information from data to code as information flows through, a system can optimize CPU resources, and bandwidth resources. This idea can be extended to optimize or prevent adverse effects from critical resources in addition to bandwidth and CPU. Memory, time of execution or buffer space could be use to trade off forms of data representation to optimize certain system parameters. The ability of data to change form within a system opens up multiple optimization paths that were previously invariant in the system. Rigorous security quantification resulting from this work allows active packets to morph by adding the required security overhead along specific communication links such that the security of the link along with the security of the morphed packet yield the proper level of security required by a given policy. Thus, security overhead is minimized.

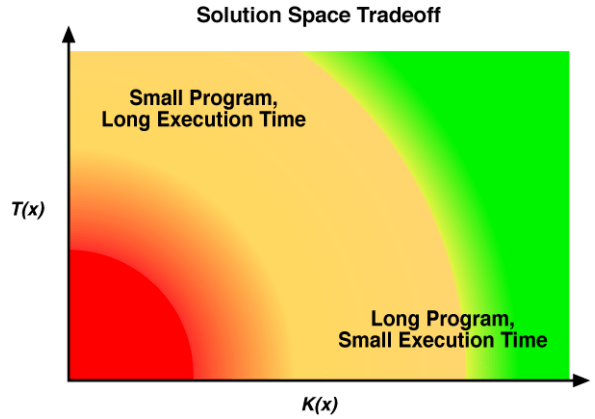

Figure 27. Program size versus Speed Tradeoffs.

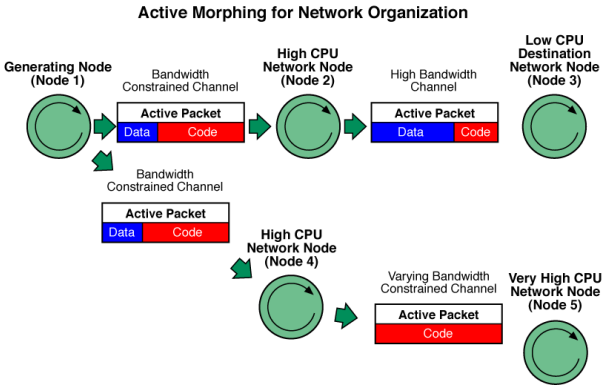

Figure 28. Active Packet Morphing for Network Optimization.

Another parameter that can optimize system resources is the knowledge of how a piece of data is used. MP3 audio is a good example of how leaving out information (specifically that which is undetectable by the human ear) can optimize data size. We introduce here the idea of "necessary" data to augment the idea of "sufficient" data or sufficient statistics that represent all information contained in the

original data. Sufficient representation of data contains all the information that the source data contains. Necessary data contains only the information that the source data contains that the destination instrument can effectively use. If you efficiently encapsulate all the information in source data in a statistical parameter you may have achieved a minimum sufficient statistic. If you further reduce this statistic such that you encapsulate only the information that is usable by the end node, you have obtained the minimal necessary sufficient statistic. Thus, Kolmogorov Complexity related ideas have tremendous impact for system optimization as well as security.

Apparent complexity

The results in Section 6.2 give an upper bound on complexity increase due to computational operations, but perhaps one can do better. In fact, the size of the shortest program one can find to produce a particular string is the best estimate for $K(x)$. Since Kolmogorov Complexity is unknowable, the best that we can do is estimate well. This introduces the idea of apparent complexity. It is as if to say "As far as I know, the complexity of this string is this." The benefit or possible way to exploit the idea of apparent complexity is that a user generating a string should have the best idea of how hard it is to generate the string. There are many reasons why a user may not choose to generate a string using the minimal size program. Perhaps a longer program can execute faster, or perhaps the generator is unknowingly using an inefficient process. However, the generator of a string of data is presumed to have knowledge of the process used to generate that data. This may in fact make the non-computability of Kolmogorov Complexity an asset: a good candidate for use in providing information assurance. The information system designer or an authorized user generating data should have better knowledge of the data process than an attacker and an attacker cannot simply compute the optimal process. Conservation of apparent complexity enables abnormalities to be tracked when the expected number of computational operations is not utilized in transforming string x into string y. Thus, even if we cannot know or compute the most efficient process for creating a string of data, we can at least gain benefit from ensuring through monitoring resources that the expected process is used. This type of assurance has in fact been used informally to detect network security problems for many years. Discrepancies in computer account charges have led to detection of attack [122]. The idea of using Kolmogorov Complexity provides the possibility of using this type of technique on a more fundamental level, where knowledge about the information content would not be required to determine unauthorized activity. The term "apparent complexity" is used to reflect the best measurement of Kolmogorov complexity available to the party undertaking the measurement.

This section addresses the question of how vulnerability relates to complexity and how this leads to the definition of a new metric called apparent complexity, AK. Figure 29 is another view of the operation of the system in Figure 8. In Figure 29, system operation, as designed by the defender, is shown as the state machine inside the smaller box in the foreground. Data of a given complexity flows in and out of the system as shown by the arrows. However, the main source of vulnerability is unknown or unexpected behavior that leads to unauthorized access of information as shown in the empty large box in the background.

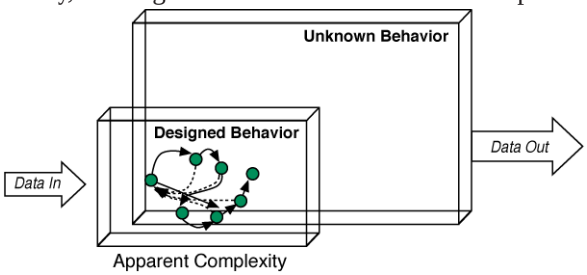

Figure 29. Vulnerability as Unknown Behavior.

Referring to our definition of attacker as scientist, the attacker can be conceptualized as developing and experimentally validating hypotheses regarding operation of the system that will lead to access to desired information. As previously discussed, the better the hypothesis, the less random, or more compact, the string representing the behavior of the system is required. Kolmogorov Complexity, K, is a measure of the smallest programmatic representation for a string that can ultimately be conceived. As the size of the attacker's representation of the string formed by observed behavior approaches K in constructing a hypothesis of system operation, the better understood the system is to the attacker. However, actually computing K is a challenge currently out of reach. Instead, a more tractable substitute is proposed called Apparent Complexity, as

shown in Definition 6.5. The salient feature of this definition is that, while similar to *K(x)*, is relative. It is relative in this definition to the capability of a user, *r*, to define the smallest program.

In this definition, if, then *r* has an unreasonable hypothesis, a hypothesis that does not contribute to an understanding of the system. If, then *r* has found a hypothesis that perfectly explains system behavior. Thus, it is reasonable to expect to lie in the range between *{K…l(x)}*. Note that *r* can be an individual, such as a single attacker or single defender, or a group with a common view of complexity. In fact, a group of individuals could converge to a common by means of a protocol similar to a routing protocol.

The string *x* in this application is a list of events by an observer external to the system, for example, the attacker or the defender. Initially the defender is likely to best understand the system. The complexity of the system, AK in Figure 29, is itself a measure of the defender's. This is because the defender's goal is usually to develop the most efficient system possible. Thus, the system is in effect the defender's best approximation of *K* with respect to generating output; in other words,. is likely to be near *l(x)* initially. Note that the attacker is likely to have, at minimum, general knowledge of such areas as login procedures and possible error conditions such as queue overflows, and will be likely to test limit points where systems interconnect as being weak links. A defender can compute, but how is determined? One method is for the defender to determine all externally observable points, and attempt to compute. The accuracy of this approximation depends upon the defender's knowledge of all the externally observable points. The accuracy also depends on the defender's knowledge being equal or better than the attacker's.

Conservation of complexity

Conserved variables enable us to deduce parameters from the presence or absence of other parameters. The Law of Conservation of Matter and Energy [127], for example, allows one to deduce how well a thermodynamic system is functioning without knowing every parameter in the system. Heat gain in one part of the system was either produced by some process or traveled from (and was lost from) another part of the system. One knows that if the thermal efficiency of a thermodynamic system falls below certain thresholds then there is problem. On the other hand, if more heat is produced by a system than expected, some unintended process is at work. A similar situation is desirable for information systems—the ability to detect lack of assurance by the presence of something unexpected, or the absence of something that is expected. This seems to be far from our reach, given that information is easily created and destroyed with little residual evidence or impact.

One possible candidate for a conserved variable in an information system is Kolmogorov Complexity. Suppose you could easily know the exact Kolmogorov Complexity $K(x)$ of a string of data, *x*. You would essentially have a conserved parameter that could be used to detect, resolve or infer events that occur in the system, just as tracking heat in a thermodynamic system enables monitoring of that system. Operations that affect string S and cause it to gain or lose complexity can be accounted for, and an expected change in complexity should be resolvable with the known (secured) operations occurring in the information system to produce expected changes in complexity. Complexity changes that occur in a system that cannot be accounted for by known system operations are indications of unauthorized processes taking place. Thus, in the ideal case where Kolmogorov Complexity is known, a check and balance on an information system that enables assurance of proper operation and detection of unauthorized activity is possible. Unfortunately, as previously discussed, a precise measure of Kolmogorov Complexity is not computable. We can, however, determine a bound on the Kolmogorov Complexity as shown in the theorems below.

Theorems of conservation

Kolmogorov Complexity, $K(x)$, can be thought of as a conserved variable that changes through computational operations conducted upon strings. In order for $K(x)$ to be a conserved variable we must be able to account for changes in $K(x)$, which must be correlated with another known value. Theorems 6.1 and 6.2 presented below enable bounds to be placed on the changes in $K(x)$ that occur due to computational operations occurring in an information system. The two theorems below show bounds on the amount of complexity that can exist due to knowledge of other strings or conducting computational operations.

While not computable from below, upper bounds on the increase in Kolmogorov Complexity can be crudely known by keeping track of the size of programs that affect data. This bound may be incredibly

loose, as it is quite possible to operate on a string and make it much less complex than the input. One would need a method to recognize this simplification. However, these results provide an intuitively attractive method for quantifying the "work" performed by a computational operation on information – the change in complexity introduced by the operation. A thorough treatment of bounds related to $K(y|x)$ and the "Information Distance" between strings is contained in Bennett et al. [122].

2.6 COMPLEXITY ESTIMATION ALGORITHMS FOR INFORMATION ASSURANCE

In order to motivate the study of complexity estimators for information assurance, this section will highlight recent results using complexity to detect FTP exploits [97] and Distributed Denial of Service (DDoS) attacks [118]. General principle and concepts of complexity based vulnerability analysis will also be discussed, providing further potential information security applications of complexity estimators.

Detection of FTP exploits using protocol header information

In [97], the use of the principle of conservation of complexity to detect anomalous use of the FTP protocol for intrusion detection is presented. Protocols enforce patterns by design, and the level of redundancy or patterns in protocol information, which can be measured through complexity metrics, was hypothesized to be an objective indication of attack vs. healthy behavior. Here complexity is estimated using Unix compress—a universal compression algorithm based on Lempel Ziv 78. TCP dump data from FTP control connections, filtered to remove certain high variance fields such as time stamps, was compressed using Unix compress and compared to trace files of healthy sessions. Results, summarized in Figure 30, indicate that attack sessions, obtained from running various FTP exploit scripts downloaded from numerous Internet sites, have measurably lower complexity (when normalized against trace length) than healthy FTP sessions.

These results indicate that the principle of conservation of complexity applied to FTP exploits enables detection of inappropriate or unhealthy use

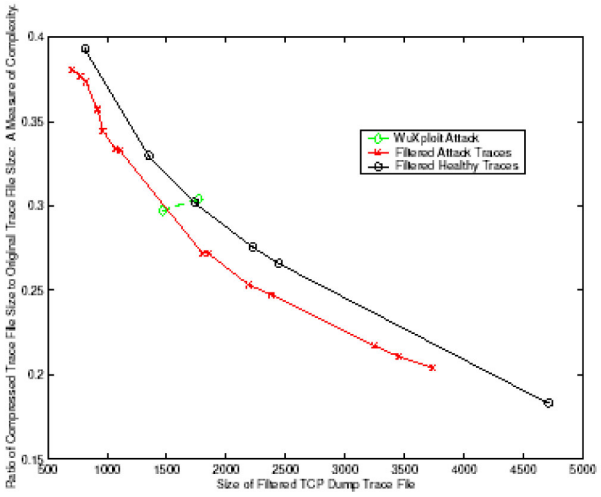

Figure 30. Inverse Compression Ratio of Filtered FTP Session Trace Files For Attacks and Healthy Sessions.

of the FTP protocol. These concepts are summarized in Figure 31.

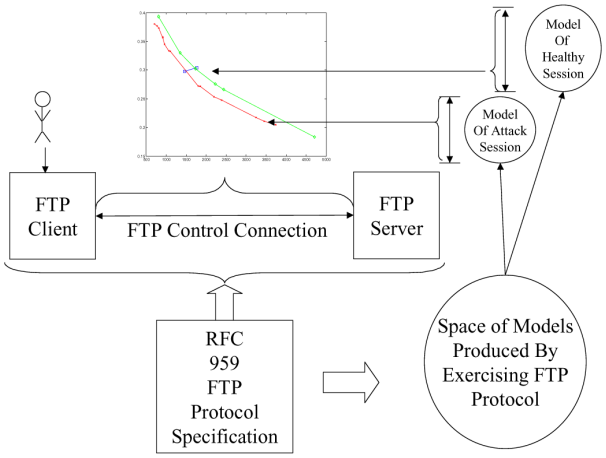

Figure 31. Conservation of Complexity Applied to FTP Exploits.

The FTP protocol specification, RFC 959, enforces and enables a certain set of behaviors that results from the rules and specifications of the protocol being exercised by the allowable space of user inputs. The large Space of Models shown in the figure indicates this set of behaviors or models. Ideally, the allowable space of models would be calculated from the protocol specifications, resulting in a space of models consisting of finite state machines, push-down automata, or whatever modeling device achieves the Kolmogorov Complexity for the particular behavior. Among this set of models would be models corresponding to healthy behaviors and models corresponding to attack behaviors. The

results in indicate that the complexity of the TCP dump header information, which can be interpreted as an objective measure of the size model represented by the data, indicates that attack behaviors (when normalized by session size) tend to be less intricate or smaller models than normal healthy sessions, thus enabling detection of exploits through complexity estimators. Further research is in progress to expand upon these results, but clearly better complexity estimators will benefit characterization of behavior models and the ability to discern attacks.

Detection of DDOS using differential complexity of data payload

Distributed denial-of-service (DDoS) attacks are caused by an attacker flooding the target machine with a torrent of packets originating from a number of machines under the attacker's control. These machines are called 'zombies'. Tools that control and launch attacks from these zombies against the target perform the attacks. The attacks can cause networks to be disabled for extended periods of time during which customers, employees, and business partners, are unable to access information or perform transactions. This section describes an approach that leverages fundamentals of information complexity to provide a flexible and effective method for detection of distributed denial of service attacks. Stated as simply and succinctly as possible, we hypothesize that information, comprising observations of actions with a single root cause, whether they are faults or attacks, is highly correlated. Highly correlated data has a high compression ratio.

The DDoS attack detection algorithm makes use of a fundamental theorem of Kolmogorov Complexity that states: for any two random strings X and Y, $K(XY) \leq K(X) + K(Y) + C$ where $K(X)$ and $K(Y)$ are the complexities of the respective strings, c is a constant and $K(X.Y)$ is the joint complexity of the concatenation of the strings. Proof for the above theorem is described in [161]. Simply put, the joint Kolmogorov Complexity of two strings is less than or equal to the sum of the complexities of the individual strings. The equivalence holds when the two strings X and Y are completely random i.e. they are totally unrelated to each other. Another effect of this relationship is that the joint complexity of the strings decreases as the correlation between the strings increases. Intuitively, if two strings are related, they share common characteristics and thus common patterns. That knowledge can be harnessed to generate a smaller program that can represent the combined string.

In terms of detection of DDoS attacks, the property given by Inequality (1) is exploited to distinguish between concerted denial-of-service attacks and cases of traffic overload. The assumption is that an attacker performs an attack using large numbers of similar packets (in terms of their type, destination address, execution pattern, timing, etc.) sourced from different locations but intended for the same destination. Thus, there is a high degree of similarity in the traffic pattern. A Kolmogorov Complexity based detection algorithm can quickly identify such a pattern. On the other hand, a case of legitimate traffic overload in the network tends to have many different traffic types. There is not much correlation between the different traffic flows and, in aggregate, the traffic appear to have a random pattern. Therefore, our algorithm samples every distinct flow of packets (distinguished by their source and destination addresses) to determine if there is a large amount of correlation between the packets in a flow. If it is determined to be so, then all suspicious flows at the node are again correlated with each other to determine that it is indeed an attack and not a case of a traffic overload.

The correlation itself is performed in the following manner. For the collected samples, the probe calculates a complexity differential over the samples. Complexity differential is defined as the difference between the cumulative complexities of individual packets and the total complexity computed when those packets are concatenated to form a single packet. If packets $x_1, x_2, x_3 \ldots x_n$ have complexities $K(x_1), K(x_2), K(x_3) \ldots K(x_n)$, then the complexity differential is computed as:

$$[K(x_1) + K(x_2) + \ldots + K(x_n)] - K(x_1 x_2 \ldots x_n)$$

where $K(x_1 x_2 x_3 \ldots x_n)$ is the complexity of the packets concatenated together as measured in a finite time

interval window (Figure 32). If packets $x_1, x_2, x_3 \ldots x_n$ are completely random, $K(x_1x_2x_3\ldots x_n)$ will be equal to the sum of the individual complexities and the complexity differential will therefore be zero. However, if the packets are highly correlated i.e. some pattern emerges in their concatenation, then the concatenated packet can be represented by a smaller program and hence its complexity i.e. $K(x_1x_2x_3\ldots x_n)$ will be smaller than the cumulative complexity.

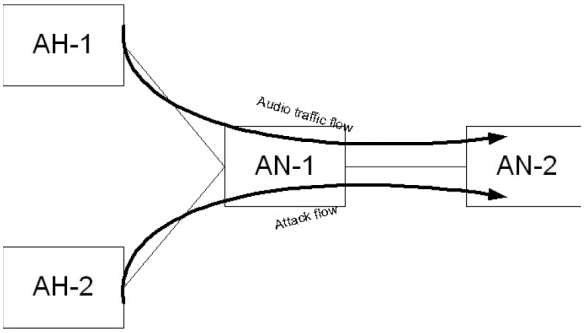

Figure 32. Topology of the experiment.

We compared our technique to a prototype packet counting algorithm for DDoS detection and found that our technique is better at discriminating traffic patterns. The experimental setup consisted of a set of active nodes arranged in the topology shown in. Node AH-1 continuously generates traffic consisting of audio packets destined for node AN-2. The load induced by this traffic is high enough that it is registered at node AN-1 as a 'suspicious' flow i.e., a traffic flow whose complexity differential exceeds the threshold. The load induced by this traffic flow is kept constant throughout the experiment. Node AH-2 generates the attack flow. The load induced by the attack flow is varied to determine the performance of the algorithms. The experiment is run twice, once with only the attack source on (node AH-2 transmitting only) and the next time with both sources on (both node AH-1 and node AH-2 transmitting). The rationale is that an attack is essentially a sustained overload induced for some time interval. The purpose of the experiment is to determine the effectiveness of the two techniques in separating and identifying an attack in the presence of background traffic

Figure 32 and Figure 33 show the performance of the packet-counting and complexity-based approaches, respectively as measured against the load induced by the two sources (in packets per second) described above. Figure 33 shows that the

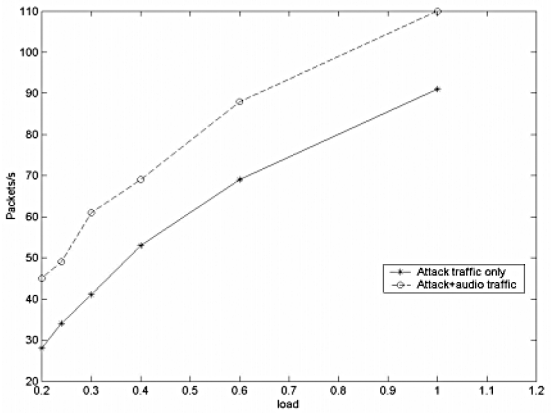

Figure 33. Performance of packet counting metric.

packet-counting metric cannot discriminate between an attack and a true overload. When the audio source is transmitting in conjunction with the attack source, any threshold set by the packet-counting algorithm running on node AN-1 will be exceeded leading to the false conclusion that the node is under attack. For example, based on the attack pattern only (dashed curve), we decide to set the threshold at 70 packets/s for a load of 0.6. When the audio source is introduced, the combined traffic trips the same threshold at a load of only 0.4, which is a false positive.

Figure 33 shows the complexity differential versus load curve for a given sampled time interval, which in this case was 10 seconds. Note that higher differential complexity corresponds to reduced complexity of the flow. In effect, the higher differential complexity estimates the deviation from the randomness inherent in a healthy network with a mix of different traffic flows. The experiment thus shows that the attack flow is estimated to be less complex over time than the ambient legitimate traffic. It is to be noted that the complexity-based metric does not change its behavior when a combination of attack and traffic sources is used. This is because the attack traffic dominates the combined flow and hence the complexity differential is roughly equal to that observed if only the attack flow existed. Therefore, the complexity-based approach is more accurate in separating false alarms from true attacks because it

can conser ve salient patterns of a traffic flow (Figure 34).

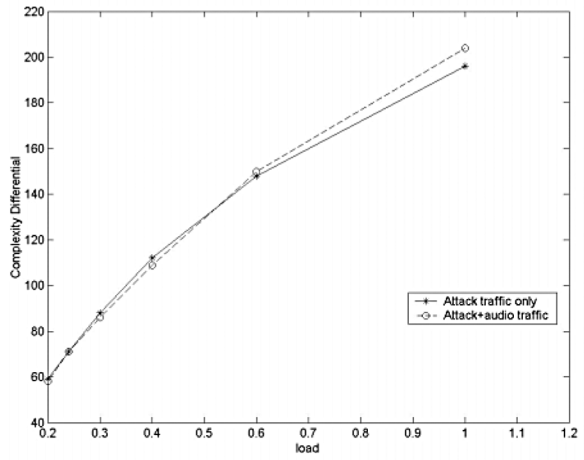

Figure 34. Performance of complexity-based metric.

Complexity-based vulnerability analysis

Complexity-based vulnerability analysis attempts to determine the likelihood of attack innovation. An attacker initially views a system as a black box. The attacker must form hypotheses about the system and test those hypotheses to successfully prosecute an attack. The hypothesis suggested by complexity-based vulnerability analysis is that less complex components of the system will be easier to understand, quicker to be manipulated, and are therefore more vulnerable. The ability of an attacker to understand, and thus successfully innovate a new attack against a system component is directly related to the size of the minimal description of that component.

A common criticism of complexity-based vulnerability analysis is that components that are more complex could be vulnerable for precisely the same reason; namely, the defender does not fully understand high complexity components, thus leaving potential vulnerabilities. However, even in this circumstance, the fundamental hypothesis remains valid, the attacker must understand components well enough to manipulate them; it is the likelihood of understanding component manipulation well-enough by the attacker that remains the object of complexity-based vulnerability analysis. In fact, the Internet Protocol suite, which claims simplicity as one of its virtues, is a popular target of attack; simplicity (the colloquial use of the term is used here because the Internet Protocol suite lacks a definition of simplicity) has not appeared to reduce its vulnerability.

The technique used by the attacker can be broadly defined as an attempt to move the system into a state unanticipated at system design time. For purposes of simplification, consider the system as described by a finite automata (FA) in a black box. The attacker can input data to the FA and receive output. The attacker can use patterns in the input/output data to deduce information about the system. The complexity-based vulnerability hypothesis can be restated more precisely as: lower complexity components of the system, from the attacker's point of view, that move the system into a state unanticipated in system design, will be deduced sooner.

The complexity-based vulnerability hypothesis is a meta-hypothesis because it is a hypothesis involving an attacker's hypotheses. The meta-hypothesis must be validated by experimentation. An ideal experiment, assuming careful setup and control is presented in[109]. The use of complexity metrics for vulnerability analysis is, along with the applications discussed above, one of the possible objective uses of complexity theory that will benefit from accurate and low overhead complexity metrics, which is the subject of this report.

Methods of estimating complexity

The previous section identifies two methods for estimating complexity – empirical entropy and universal compression algorithms. Both metrics are related to Kolmogorov Complexity, in that K(x) is the ultimate compression bound for a given finite string x. Thus any universal compression algorithm is a natural choice for a complexity estimator. However, since universal compression algorithms are designed to apply to populations of strings, the ultimate compression bound for a specific string is generally smaller than that achieved by a universal compressor.

Figure 35 describes how the quality of complexity

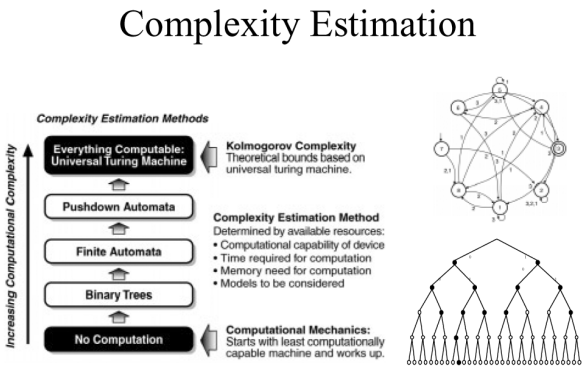

Figure 35. Hierarchy of computational platforms in estimating complexity.

estimator is tied to the computational model considered in estimating complexity. Simple estimates, such as empirical entropy lie at the bottom of Figure 35 and can be implemented using very simple computational platforms with very little overhead. Popular universal compression algorithms can be implemented with context free grammars (finite automata) and provide additional accuracy in complexity estimation at additional computational expense. Kolmogorov complexity is the theoretical limit for complexity estimation that requires computational capability equal to that of a universal Turing machine on which anything computable can be computed. Due to the halting problem in searching for the theoretical limit for a specific string (we cannot determine if a candidate compressor will ever halt) the theoretical bound cannot be achieved One example of a universal compression algorithm is the universal compression algorithm designed by Lempel and Ziv which is known as LZ78 [183]. The LZ78 algorithm partitions a string into prefixes that it hasn't seen before, forming a codebook that will enable long strings to be encoded with small indexes (Figure 36).

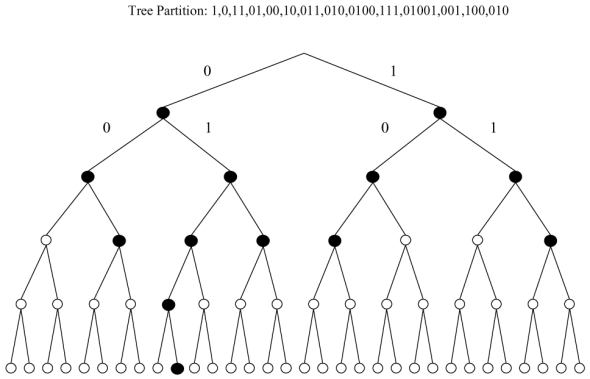

Figure 36. LZ78 binary tree representation of the partition for the binary string: 1011010010011010010011101001001100010. Nodes contained in the partition are colored in black.

Consider an example to illustrate how this algorithm works: LZ partitioning of the string:

1011010010011010010011101001001100010

is performed by inserting commas each time a sub-string that has not yet been identified is seen. The following partition results:

1,0,11,01,00,10,011,010,0100,111,01001,001,100,010

Figure 36 represents this string partition as a binary tree. The nodes marked in black of the five level tree shown are nodes contained in the LZ78 partition of the example string. Nodes that are not filled in indicate code words or phrases that are not contained in the LZ78 partition. Each node or phrase occurs exactly once in the string with the exception of the last phrase which may be a repeat of a previously seen node. Good compression (low complexity estimation) results when the LZ78 partition contains a deep, sparse tree, while poor compression (high complexity estimation) results from strings that are less deep and more completely populated at each level

A comparison of ubiquitous complexity estimators

This section compares the performance of the complexity estimators used in our work so far, viz. empirical entropy, Zlib-compress and LZ78-code-length estimator.

Empirical entropy estimation technique measures the weight of '1's that occur in a binary string in relation to the length of the string. Thus a string with a

higher number of 1's is estimated to have a higher complexity as compared to a string of the same length with a lower number of 1's. This estimation technique is computationally simple and fast but not very accurate.

Zlib-compress uses the java.util.zip.Deflater class found in the Java compression library [184]. Attacks using Kolmogorov Complexity Metrics. The inverse compression ratio is estimated to the complexity of the string using this method. This universal compression algorithm utilizes LZ77 coupled with Huffman coding.

The LZ-code-length estimation method attempts to guess at the amount of compression possible for a string using the LZ78 compression algorithm without actually performing it. The three estimators were compared for accuracy by conducting the following experiment. A byte buffer was filled with partitioned into two, one of which was filled with patterned data (i.e., data having a known pattern) and the other part was filled with random data. The estimators were run on the buffer to get a complexity value for each estimator. The ratio of the random data to the pattern data in the buffer was increased in each successive run set. Thus the first set had all pattern data in the buffer (and thus low complexity) while the final set had all random data (and thus high complexity). The pattern in the patterned data was varied to prevent bias and the average complexity value was chosen for each set.

Figure 37 shows a comparison of the three estimators. Zlib-compress has a linear change in complexity as the ratio of non-pattern data to pattern data increases and is the most accurate of the three estimators. LZ-encode is seen to overestimate the complexity of the data with respect to Zlib-compress throughout the range of the experiment. Empirical entropy is seen to be the least accurate over the range. It grossly overestimates the complexity when there is more pattern data in the buffer because it is simply counting the weight of the 1's in the string instead of looking for patterns.

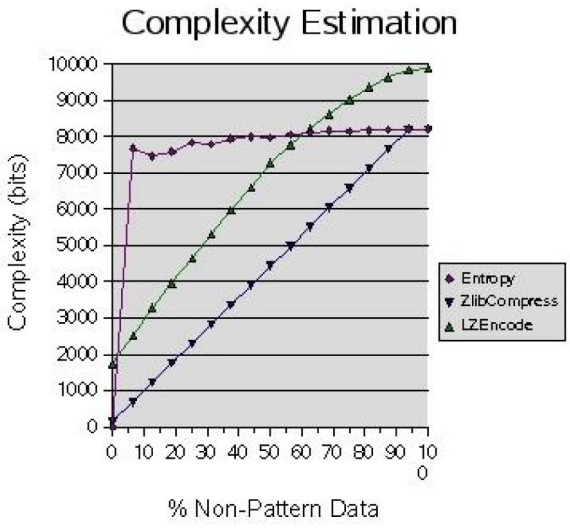

Figure 37. Complexity estimate in bits vs. randomness of data.

Figure 38 shows the variance of the estimated values over the range of the experiment. Zlib-compress can be seen to be least sensitive to the type of pattern data used. Empirical entropy estimation variance is highest when different patterns are used for the same ratio of random to pattern data. This is because the count of 1's in the pattern data affects the estimation value. On the other hand, when the data is more random, the ratio of 1's is expected to be remain close to 0.5.

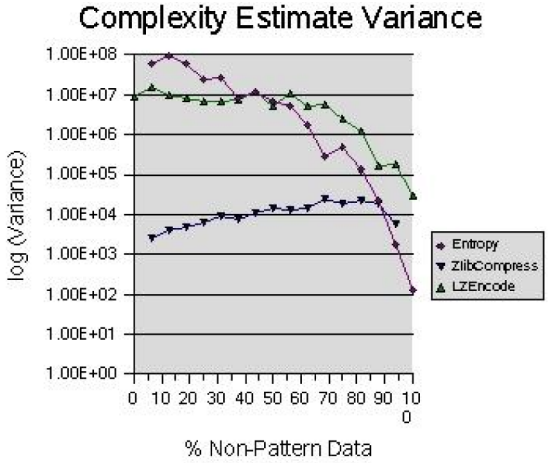

Figure 38. Complexity estimation variance vs. randomness of data.

Minimum description length principles

The previous section identifies the challenges in dealing with variance and accuracy of complexity estimators. We will now address a related concept which will be used to build a new complexity estimation algorithm Two statistical application techniques for inductive inference that are quite similar to the Kolmogorov Complexity, and with strong links to information theory are known as Minimum Message Length (MML) coding and Minimum Description length (MDL) coding [181]. For our purposes these techniques are equivalent and will be used inter-

changeably. MML coding encodes information as a hypothesis or model that identifies the presumptive distribution, from which data originated, appended with a string of data, coded in an optimal way. The total descriptive constant C of a string under the concepts of MML or MDL string can be defined in two parts as $C = M + D$, where M is the model cost and D is the data cost. In the preferred two-part description the model M describes all regularity associated with a string and the data portion D describes the random elements of the string where the sum of the lengths of these two parts is equal to the Kolmogorov complexity of the string. In other words, the best two-part description of the data should not be longer than the optimal single part description of the data. A two-part description is known to exist if only consisting of a set containing a single string. There are generally many two-part descriptions of a string, the shortest being termed the algorithmic minimum sufficient statistic. A thorough treatment of algorithmic statistics for the class of models consisting of finite sets and probability distributions is contained in [172].

Sophistication

One criticism of the use of Kolmogorov Complexity for characterization of information is that it is in some sense a measure of randomness. Random information is not necessarily important information. This criticism can be addressed by thinking of Kolmogorov complexity as two parts. Sophistication is a measure of meaningful information that was formalized in [183], harnessing the fact that the shortest description of an object (with length equal to Kolmogorov Complexity) can be expressed in two parts. The first part describes a Turing machine that, given the second part as input, produces a given string. The first part of the code models the regularities of the string, while the second part describes the irregularities. The combination of model and code that construct a string must together be the smallest under the Minimum Description Length criteria. Vitányi in [183] expands the model space to include not just finite sets but also any computable model in the recursive function class. The relationships between Sophistication and Kolmogorov Complexity are shown in Figure 39.

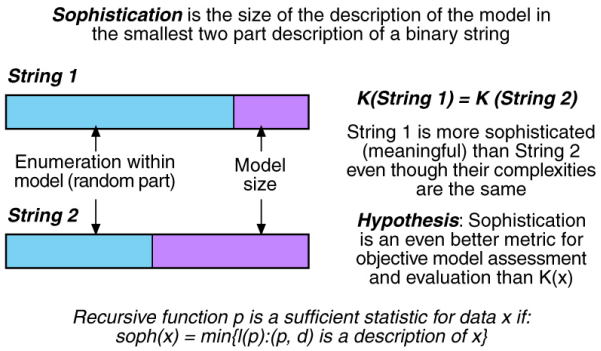

Figure 39. Comparison of sophistication and Kolmogorov Complexity.

The motivation for studying sophistication is rooted in a search for a method to quantify "meaningful information". In an attempt to rigorously define the normally subjective notion of "meaning", Vitányi associates the model with the meaningful information involved in the string, while the second part of the code is the meaningless part. For example, a purely random string of length L has Kolmogorov Complexity on the order of L. We could compare this to a string taken from a piece of English text that has Kolmogorov Complexity of L (the uncompressed string would have length much larger than L). Even though these two strings have similar complexity, there are large differences between their two-part codes. The random string has no model associated with it and is essentially all data. The English text will conform to some model associated with the frequency of use of letters, words and phrases. If the author of the text is known, a better model tuned to the writer's vocabulary is possible. Thus, minimal two-part code of the sample of English text will consist of a fair amount of model in addition to data encoded under the model. Thus the English text sample would be considered more sophisticated than the random data even though they are equally complex.

A new complexity and sophistication estimation algorithm

Sophistication motivates the search for new complexity and sophistication metrics that not only indicate the compressibility or randomness of a given string, but also indicate information about the size of the model that could produce the string. Universal compression algorithms are not designed to do this, since their ultimate goal is to produce a one

part encoded version of a string that can be reconstructed into the string at will. In this section, we derive a heuristic that will lead to a new universal sophistication and complexity estimator that forms a two-part code for a given string.

We consider compression of a finite binary string X of length L. We seek the optimal partition of X into I symbols that can be encoded using a near optimal encoding strategy such as Huffman coding such that the combination of the descriptive cost of the model plus the encoding of the data under the model are minimized. The following parameters are defined:

Table 3 Table 1: OSCR parameters

Parameter	Meaning
L	Length of finite string S
I	Total number of symbols in partition $i \in [1,I]$
l_i	Length of symbol i
r_i	Number of repetitions of symbol i in S If repetitions for all symbols are equal then $r_i = r$ and $R = \sum_i r_i$
R	Total number of repetitions,
L_i	Length of S consumed by symbol i $L = l_i r_i$ $L = \sum_i L_i$
C_p	Total Descriptive cost of S under partition p. Equal to the sum of the model description M plus the encoding of the data under the given model.
c_i	Descriptive Cost of Symbol i. This parameter will be derived in section 4
D_i	Symbol Compression Ratio (SCR)

The effect of a partition on MML

The entropy of a distribution of symbols (H_s) defines the average per symbol compression bound in bits per symbol for a prefix free code. For a distribution p of I symbols:

$$H_s = -\sum_i p_i \log_2(p)$$

Huffman coding and other strategies can produce an instantaneous code approaching the entropy when the distribution p is known. But what is the best encoding possible when the source distribution is not known? One way to proceed is to measure the empirical entropy of the string, that is the entropy defined inherently by the input string itself. However, empirical entropy is a function of the partition and depends on what sub-strings are grouped together to be considered symbols. See for a consideration of some of the inadequacies of the well-known Lempel-Ziv algorithms in dealing with higher order empirical entropies.

Our goal is to optimize the partition (the number of symbols, their length, and distribution) of a string such that the compression bound for an instantaneous code, which is equal to $R*H_s$, plus the codebook size is minimized according to the MML criteria. We estimate the codebook size (model descriptive cost M) to be the sum of the lengths of unique symbols:

$$M = \sum_i l_i$$

Thus we estimate the total descriptive cost C_p:

$$C_p = M + R \cdot H_s$$

Consider for now that all symbols are equally likely and of equal length. Thus:

$$r_i = \frac{R}{I} = r, \quad l_i = \frac{L}{R} = l \text{ and } \quad H_s = \log_2(I)$$

describes how entropy changes as unique symbols are added to the partition; each added symbol increasing the number of bits required to encode each symbol in a less than linear fashion. This increased descriptive cost per symbol must be traded against symbol length and number of repetitions.

For a given number of unique symbols, more repetitions will at first tend to decrease the overall description length, since the fixed length string of size L will now be divided into shorter words of size l and the codebook for the string will now be shorter to describe.

The description length decreases (the reduction in model size dominates) until a minimum occurs where the benefit from a decreased codebook size is offset by the fact that more symbols of a fixed average encoded length (on the order of H_s) must be appended to the description. Figure 31 plots description length vs. number of repeats for various size equally likely alphabets based on a 1024 bit string. The knee in the curve for each number of symbols represents the optimal number of repetitions for a certain symbol alphabet size.

The minimum for a given number of symbols in the data can be calculated as follows:

$$C_p = \sum_i l_i + R \log_2(I) = I\frac{L}{R} + R \log_2(I)$$

where s is the number of symbols contained in the data (Figure 40).

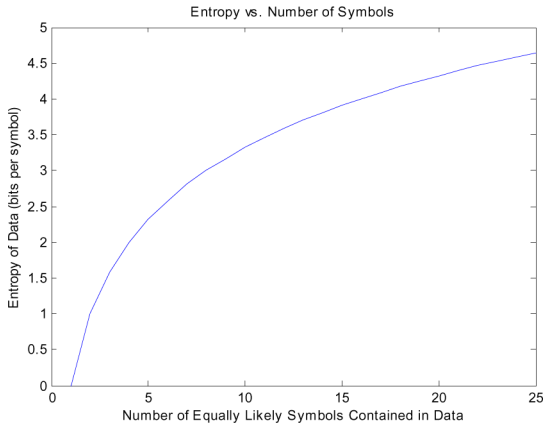

Figure 40. More equally likely symbols in a partition cause the Entropy to increase – raising the bits per symbol descriptive cost in a less than linear manner.

It can be easily shown that for a given number of symbols I, a minimum description length can be expected if each equally likely symbol is repeated r times where:

$$r = \frac{1}{I}\sqrt[I]{\frac{IL}{\log_2(I)}}$$

This number represents the optimal tradeoff between codebook size and encoded data size for a given string partition into equally likely, equal length symbols (Figure 41).

As shown in Figure 41, the benefit of repeated patterns in fixed size data is overcome more quickly in a large equally likely alphabet. For the case where the entropy is less than one (i.e., there is only a single symbol) the benefit increases until all symbols are repeats, as expected. As an example of this principle, consider a 1000 bit string

X=10101010101010101010101010101010101010101…

There are many ways to parse this string. As shown in Figure 42, in an optimal parsing of this string with a two symbol, equal length, and equally likely alphabet, each symbol would repeat about 20 times. The optimal descriptive cost for this partitions is approximately 100 bits = 25 bits to describe each

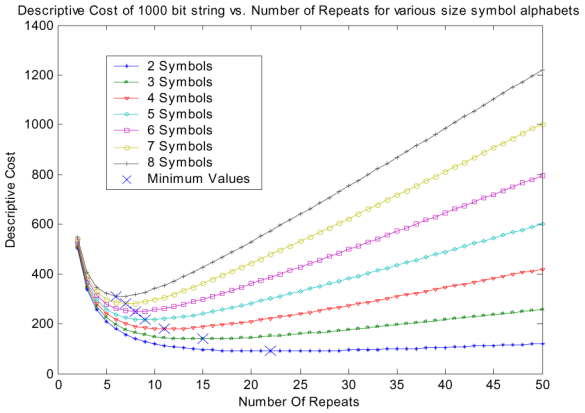

Figure 41. Symbol length and number of repetitions of equal length equally likely symbols comprising a string of finite length produce competing affects in total string descriptive cost.

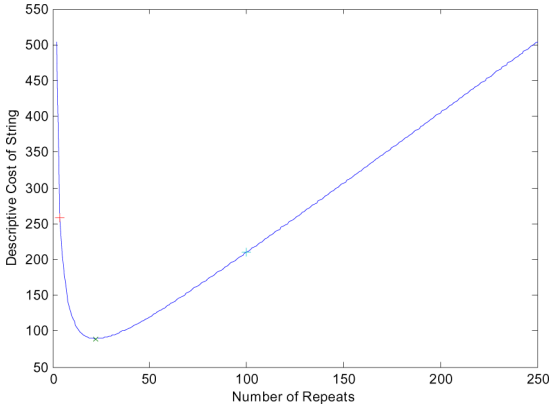

Figure 42. Descriptive cost vs. number of repeats for two symbol partitions of the 1000 bit string 101010101….

codeword, 40 bits to encode each codeword in the string. Note that we have ignored "Comma" cost, which will be required to show separation of code words in the codebook. The codebook listed below achieves this partition:

A=1010101010101010101010101

B=0101010101010101010101010,

with the encoded string represented by ABABABAB… 20 times.

Two other partitions are noted in, both with similar code words except the length of each codeword is different. In the first case, (represented by a + in Figure 42) each symbol is 125 bits long and repeated four times. In the second case each symbol is 5 bits long and repeated 100 times. Both cases require a

much lar ger descriptive cost for the string. In the first case the additional burden is on the codebook that must describe two 125-bit symbols. In the second case the burden is on the encoding of the data, which must describe 200 symbols once encoded. As a benchmark, the LZ78 can encode this string in 378 bits.

The above analysis shows that when partitioning a string both length of symbols and number of repetitions of symbols must be traded off and optimized in order to minimize descriptive length. We develop a method to treat non-uniform distributions in the next section.

Symbol compression ratio

In seeking to partition the string so as to minimize the total string descriptive length C_p, we consider the length that the presence of each symbol adds to the total descriptive length and the amount of coverage of total string length L that it provides. As described in Section 3, the descriptive cost of the model is on the order of the sum of the lengths of unique symbols in the partition. The descriptive cost of the encoded data is on the order of $R{\cdot}H_s$, where H_s is the entropy of the symbol partition and R the total number of symbols in the string. The probability of each symbol, p_i, is a function of the number of repetitions of each symbol:

$$p_i = \frac{r_i}{R}$$

Thus, we have:

$$H_s = -\sum_i p_i \log_2(p_i) = -\sum_i \frac{r_i}{R} \log_2\left(\frac{r_i}{R}\right)$$

Since

$$R = \sum_i r_i$$

we can simplify this to:

$$H_s = \log_2(R) - \frac{1}{R}\sum_i r_i \log_2(r_i)$$

H_s is a measure of the ability to encode the distribution p of I symbols. The smaller H_s the fewer bits per symbol required to encode each symbol. For a fixed I, H_s is maximized when all I symbols are equally likely. Total number of symbols R multiplied by H_s will yield the length required to encode the data using an optimal technique. This can be added to the codebook size to achieve a bound for the descriptive complexity under partition p (Figure 43).

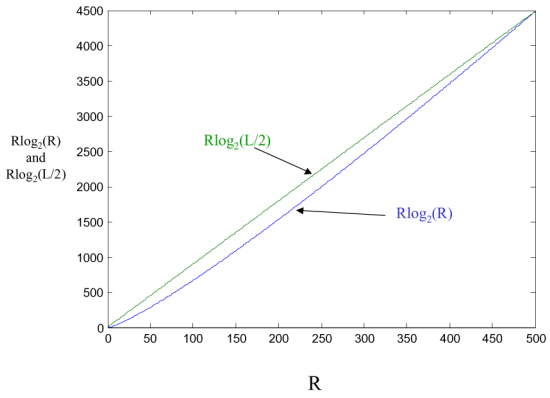

Figure 43. Estimate of RIog2(R) .

Thus, descriptive length of the string under partition p is equal to:

$$C_p = R\log_2(R) + \sum_i l_i - r_i \log_2(r_i)$$

Our goal is to find a per symbol descriptive cost. In the equation above, all terms are defined per symbol i with the exception of the first term. We define the relation:

$$R\log_2(R) = \sum_i r_i \log_2(R) = q\sum_i r_i$$

where q is a constant estimating $log_2(R)$. For R that can vary between 2 and $L/2$ for symbols of size 2 bits or greater, $log_2(R)$ can be estimated to enable an incremental, per symbol formulation for C_p. Estimating q:

$$q = \log_2\left(\frac{L}{2}\right)$$

results in a conservative approximation for $Rlog_2(R)$ over the likely range of R as shown in for partitions of strings having length equal to 1000 bits. The per-symbol descriptive cost can now be formulated:

$$c_i = r_i\left[\log_2\left(\frac{L}{2}\right) - \log_2(r_i)\right] + l_i$$

$$D_i = \frac{c_i}{L_i} = \frac{r_i\left[\log_2\left(\frac{L}{2}\right) - \log_2(r_i)\right] + l_i}{l_i r_i}$$

C_p is less than the sum of the individual code words due to the conservative approximation for $\log_2(R)$. A lower bound on the estimate for d can be formed as well to create upper and lower bounds on the descriptive length C_p:

$$c_{min_i} = r_i \log_2(r_i) + l_i - r_i \log_2(r_i) = l_i$$

$$\Rightarrow \sum_i c_{min_i} \le c_p \le \sum_i c_i$$

This bound can be tightened, as better estimates of the total number of repetitions in a partition become known.

We now have a metric that conservatively estimates the descriptive cost of any possible symbol in a string. A measure of the compression ratio for a particular symbol is simply the descriptive length of the string divided by the length of the string "covered" by this symbol. We define the compression ratio of a symbol (SCR) to be (Figure 44):

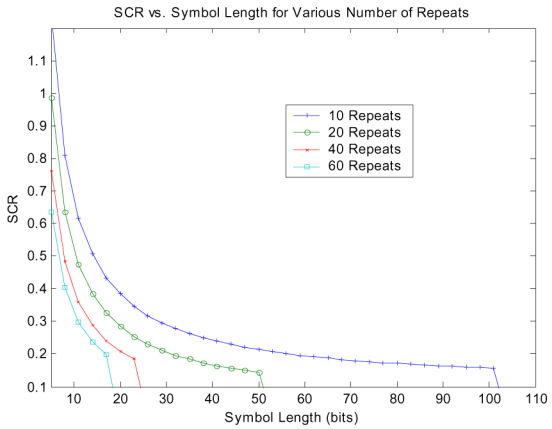

Figure 44. SCR vs. Symbol Length for 1024-bit String.

Thus we have a metric to describe the effectiveness for compression of a particular candidate symbol in a possible partition of a string that can be used for comparison in forming a partition Examining SCR above it is clear that good symbol compression ratio arises in general when symbols are long and repeated often. Clearly, selection of some symbols as part of the partition is preferred to others.

Figure 44 and Figure 45 show how symbol compression ratio varies with the length of symbols and number of repetitions for a 1024 bit string. In both figures the discontinuities reflect when symbol length times number of repeat exceeds string length, and SCR is therefore undefined (Figure 45).

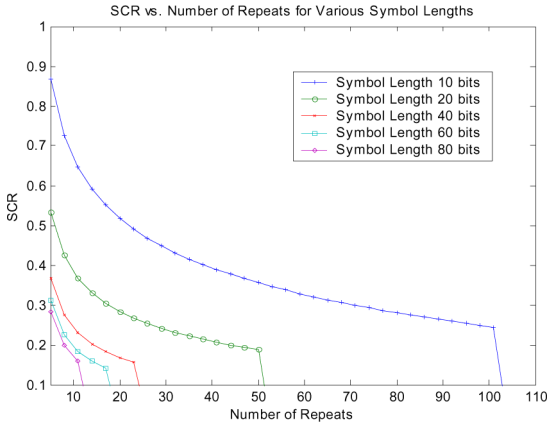

Figure 45. Figure 16 SCR vs. Repeats for 1024 bit String.

Optimal Symbol Compression Ratio (OSCR) algorithm

The Optimal Symbol Compression Ratio (OSCR) algorithm forms a partition of string x into symbols that have the best symbol compression ratio among possible symbols contained in *x*. The concept is to form a codebook dictionary that provides near optimal compression by adding one codeword at a time based on the code words symbol compression ratio. The algorithm is shown in the sidebar.

OSCR ALGORITHM

1. Form a binary tree of all non-overlapping sub-strings contained in x that occur' 2 times and note the frequency of occurrence.
2. Calculate the SCR for all nodes (sub-strings). Select the sub-string from this set with the smallest SCR and add it to the model M.
3. Replace all occurrences of the newly added symbol with a unique character to delineate this symbol. Repeat steps 1 and 2 with the remaining binary string elements until no binary elements remain.
4. When a full partition has been constructed, use Huffman coding or another coding strategy to encode the distribution, p, of symbols.

The following comments can be made regarding this algorithm:

- This algorithm progressively adds symbols that do the most compression "work" among all the candidates to the code space. Replacement of these symbols left-most-first will alter the frequency of remaining symbols.
- SCR is at first based on the crude estimate for *R* discussed previously. Justification of this estimate and possible iteration of the algorithm can be achieved by performing the algorithm again upon completion with the computed value for *R* defined by the partition calculated by the algorithm. An unchanged partition validates the output of the algorithm. If the partition does change, the algorithm can be iterated using computed values of R until it converges or repeats a previous case.
- A less exhaustive search for the optimal SCR candidate is possible by concentrating on the tree branches that dominate the string.
- The algorithm does not require a prefix free partition of the string. The left-most substitution of highest SCR symbols first suffices to produce a unique partition. A prefix free code is however assumed in the encoding.
- Reduction of the binary tree size can be achieved by noting minimum SCR at each level and considering bounds from tree nodes (Figure 46).

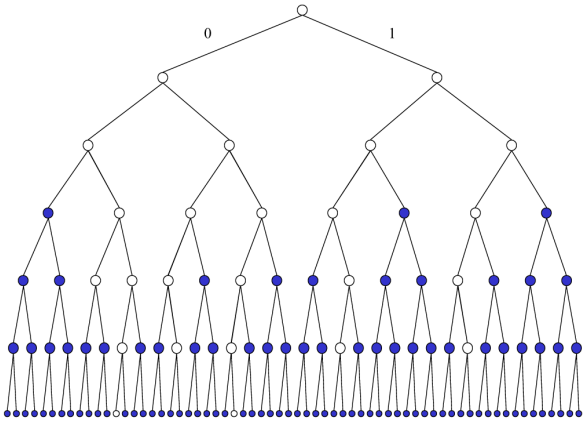

Figure 46. Binary Tree for a specific string. Nodes included are in white.

As an example consider the 40-bit string below:

X = 0011001001000010100101001100100110011001.

A full five level binary tree expansion of all sub strings contained in this string does not include all possible nodes. Rather, only certain possible patterns are contained in any partition. identifies the possible sub strings that occur. The first pass of the OSCR algorithm will produce the binary tree of symbol frequencies shown in Figure 47. Note, in building this tree we noted and utilized the fact at the second level of the tree the SCR of 12 repetitions of the symbol 01 is < 0.5, thus we did not expand tree nodes with two or less repeats.

As shown in Figure 47, the symbol 001 is repeated 10 times and has the smallest symbol compression ratio. Substituting "A" for this symbol produces the string:

X' = A1AA00A01A01A1AA1A1A (Figure 47)

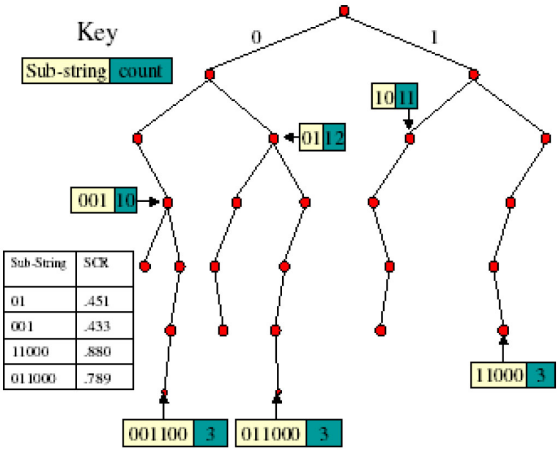

Figure 47. Binary Pattern Tree in first pass of algorithm.

Iterating the algorithm shows that the second symbol candidate, 01 that has the smallest compression ratio, does not promote compression. Thus the remaining symbols simply substitute for 1 and 0:

ABAACCACBACBABAABABA

This provides the distribution of symbols shown in Figure 4. The entropy of this symbol distribution is 1.48 bits per symbol. This can be approximated by the Huffman tree shown in Figure 48, which

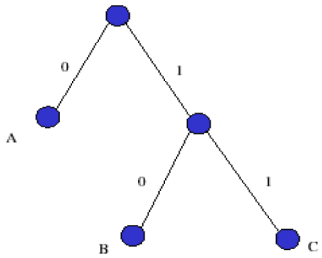

Figure 48. Huffman Tree for Symbol Partition.

achieves an expected encoded length of 1.5 bits per symbol.

Table 4 Symbol Distributions

Substring	Symbol	Probability	New Code
001	A	0.5	0
1	B	0.3	10
0	C	0.2	11

Simple substitution of the new code results in the encoded string:

X' = 0100011110111001110010001001000:

Thus, the encoded message has been reduced to 30 bits from the original 40 bits. The descriptive cost of the codebook is estimated as the sum of the lengths of symbols, which is equal to 5 bits. Depending on the strategy for delineating the separation between code words and defining the prefix free encoding of the codebook this descriptive cost could vary. Estimated results compared with LZ78 for several short strings are shown in Table 5.

Table 5 Encoded Lengths for several short strings.

String	LZ78	OSCR	Model
010010110100101101001011O	40	0.20	010,11,0
101010101010101010101010101	30	12	1010,1
111011011110110111011101	30	20	101,11,1

The previous example illustrates the concept of the OSCR algorithm. As is the case with Lempel-Ziv and other compression algorithms, greater compression is realized on strings of longer length. In addition to compression, the algorithm provides the following benefits:

- The model developed can be used to produce a typical set of strings to which x belongs.
- The symbol alphabet size of 3 symbols is an inherent parameter associated with this string that can be used to compare it with other strings. The symbol size measurable parameter related to complexity that reflects the number of variables address by the string.

Comparison with Lempel-Ziv78

The OSCR and LZ78 algorithms share the approach of dictionary coding strategies, achieving compression through giving smaller representations for longer repeated strings. The difference is that the string patterns identified and indexed in LZ78 are precisely the unique string patterns that occur from left to right that have not been seen before. No effort is made to construct a partition of repeated patterns that gives a more optimal encoding than that which falls out of the patterns or string phrases that occur first. In most implementations all substrings are given an equal size codeword (index), therefore a frequently occurring short codeword may actually be expanding the size of the encoded string. The benefit of the Lempel-Ziv approach is the computational simplicity and ease with which the dictionary or codebook is communicated. The dictionary is essentially interleaved in the encoded data and commas or explicit communication of the codebook is not required.

The OSCR takes the other extreme by identifying the repeated patterns that contribute most to compression of the string at the expense of computational requirements. One can envision a combination of these two philosophies that will be addressed in future work that provides a continuum of gradation between compression gain and computational requirements.

Comparison of estimators for detection of FTP exploits

The goal of the OSCR algorithms is to improve complexity estimation in a manner that provides the ability to discern attack vs. healthy behaviors. Results from Figure 172indicate a separation of curves for attack vs. healthy ftp traffic. Widening these curves will result in better ability to discern exploits with fewer false alarms.

Figure 49 shows the difference in complexity estimation provided by various complexity estimators healthy session and an attack session trace files of about 2kbits. As shown in the figure, empirical entropy and straight LZ78 estimation incorrectly discern healthy from attack behavior. OSCR widens the curve over Zip compress (Zlib), providing a better margin for error in discerning attack, despite the fact that Zip compress provides a better compressor

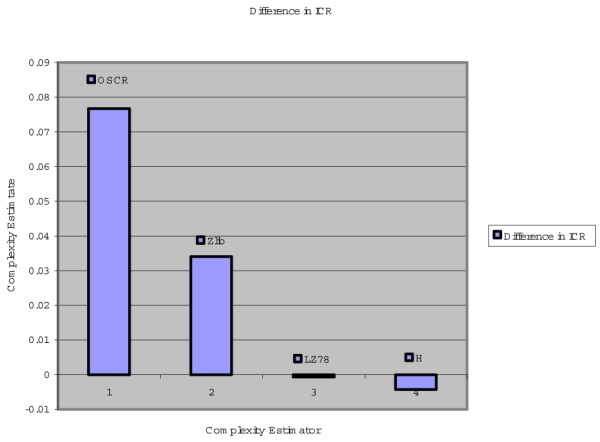

Figure 49. Comparison of OSCR vs. Zip Compress for FTP data.

as shown in Figure 50. Further data must be taken to validate this gain. Additional gains in OSCR compression can be made through optimizing the model cost beyond simple sum of the codeword length (Figure 50).

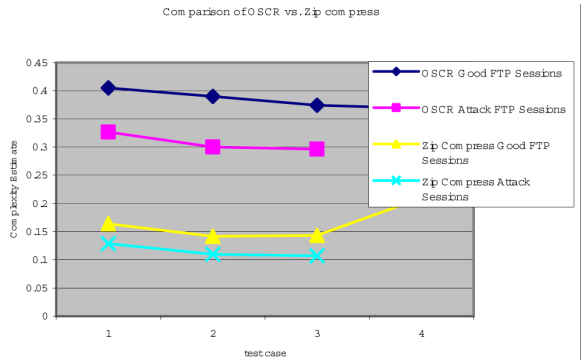

Figure 50. Comparison of OSCR vs. Zip Compress Compression Ratio.

2.7 DETECTING DISTRIBUTED DENIAL-OF-SERVICE ATTACKS USING KOLMOGOROV COMPLEXITY METRICS

Distributed denial-of-service attacks are caused by the attacker flooding the target machine with a torrent of packets originating from a number of machines under the attacker's control. These machines are called 'zombies'. The attacker typically uses ICMP or UDP packets for the attack. Typical detection techniques [156, 157] for these types of attacks rely on filtering based on packet type and rate. Essentially, the detection software attempts to correlate the type of packet used for the attack, be it ICMP or UDP, with the destination. While these techniques have reasonable success, they are not very flexible. For example, these techniques will fail if a new type of packet is used for attack or if the attack consists of a traffic pattern that is a combination of ICMP and UDP packets. In such cases, packet profiling is defeated. This report describes an approach based on fundamentals of information complexity that is both flexible and effective.

Stated as simply and succinctly as possible, we hypothesize that information, comprising observations of actions with a single root cause, whether they are faults or attacks, is highly correlated. Highly correlated data has a high compression ratio. The Kolmogorov Complexity, *K(x)*, of a string of data measures the size of the smallest program capable of representing the given piece of data [10]. It measures the degree of randomness for the given data. The length of the shortest program to generate a completely random string is equal to the size of the string itself. For all other cases, it is smaller than the size of the string and the program size becomes smaller as more regularity or pattern is discernible from the string. A side effect of this measure is its ability to represent the correlation between disparate pieces of data. This side effect is exploited to design an effective method for detecting DDoS attacks (Figure 51).

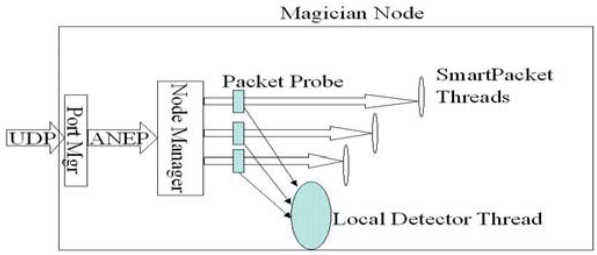

Figure 51. Implementation in Magician Active Node.

Approach

The DDoS attack detection algorithm makes use of a fundamental theorem of Kolmogorov Complexity that states: for any two random strings X and Y,

$$K(XY) <= K(X) + K(Y),\ldots\ldots\ldots\ldots(1)$$

where *K(X)* and *K(Y)* are the complexities of the respective strings and *K(XY)* is the joint complexity of the concatenation of the strings. Simply put, the joint Kolmogorov complexity of two strings is less than or equal to the sum of the complexities of the individual strings. The equivalence holds when the two strings *X* and *Y* are totally random i.e. they are completely unrelated to each other. Another effect

of this relationship is that the joint complexity of the strings decreases as the correlation between the strings increases. Intuitively, if two strings are related, they share common characteristics and thus common patterns, That knowledge can be harnessed to generate a smaller program that can represent the combined string.

The concept of "Conservation of Complexity" was introduced in [210]. This concept relates to the ability to discern an attack by monitoring the complexity change due to processes occurring in the system and imposing bounds that identify unauthorized processes—noted by complexity changes that are either too great or too small to be from authorized processes. Figure 52 describes the

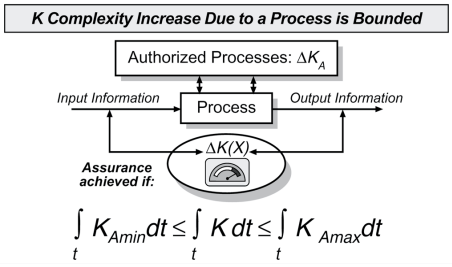

Figure 52. Principle of conservation of complexity.

concept of conservation of complexity. This concept was first applied to a closed system, where the processes are known and able to be monitored by complexity probes. In the distributed case of a denial of service attack, the process is not known, but bounds on the differential complexity allowed by the distributed processes are still able to be enforced.

The above given by inequality (1) is exploited to distinguish between concerted denial-of-service attacks and cases of traffic overload. The assumption is that an attacker performs an attack using large numbers of similar packets (in terms of their type, destination address, execution pattern etc.) sourced from different locations but intended for the same destination. Thus, there is a lot of similarity in the traffic pattern. A Kolmogorov complexity based detection algorithm can quickly identify such a pattern. On the other hand, a case of legitimate traffic overload in the network tends to have many different traffic types. The traffic flows are not highly correlated and appear to be random. Therefore, our algorithm samples every distinct flow of packets (distinguished by their source and destination addresses) to determine if there is a large amount of correlation between the packets in a flow. If it is determined to be so, then all suspicious flows at the node are again correlated with each other to determine that it is indeed an attack and not a case of a traffic overload.

The architecture for DDoS detection has been implemented in an active network for ease of deployment and flexibility in testing. As shown in Figure 53, it consists of a packet complexity probe

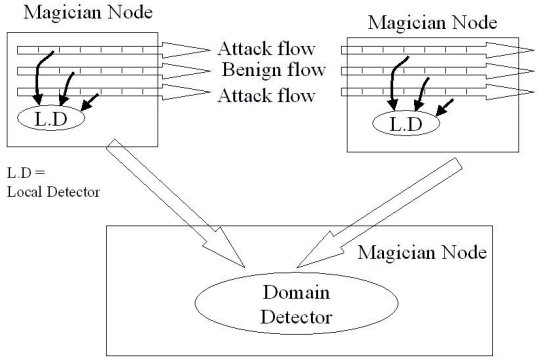

Figure 53. DDoS detection architecture.

(described in detail in the next section) associated with every traffic flow through a node that periodically samples packets in the flow. For the collected sample, the probe calculates the complexity differential for the sample. ***Complexity differential*** *is defined as the difference between the cumulative complexities of individual packets and the total complexity computed when those packets are concatenated to form a single packet.* If packets $x_1, x_2, x_3,\ldots,x_n$ have complexities $K(x_1)$, $K(x_2)$, $K(x_3)$,…, $K(x_n)$, then complexity differential is computed as:

$$[K(x_1) + K(x_2) + K(x_3) +\ldots+ K(x_n)] - K(x_1x_2x_3\ldots x_n),$$

where $K(x_1x_2x_3\ldots x_n)$ is the complexity of the packets concatenated together. If packets $x_1, x_2, x_3,\ldots,x_n$ are completely random, $K(x_1x_2x_3\ldots x_n)$ will be equal to the sum of the individual complexities and the complexity differential will therefore be zero. However, if the packets are highly correlated i.e some pattern emerges in their concatenation, then the concatenated packet can be represented by a smaller program and hence its complexity i.e., $K(x_1x_2x_3\ldots x_n)$ will be smaller than the cumulative complexity. In effect, we use the measure of the compressibility of the packets accumulated in a given time interval to determine correlation. If the complexity differential is greater than a preset threshold for the flow, the flow is marked as suspect and the collected sample is referred to a Local Detector running on the node.

The Local Detector receives all such samples from various suspicious flows and correlates all the samples together using the same complexity differential calculation. If there is only one suspect flow, no correlation is performed. If the complexity differential again exceeds the threshold, all suspect flows (including the case of a single flow) are referred to a Domain Detector that is running on some other node on the local network domain. This hierarchy of detectors cooperates to detect distributed denial of service attacks in the network itself. This hierarchy is shown in Figure 53.

Complexity estimates

While it is known that, in general, Kolmogorov complexity is not computable, various methods exist to compute estimates of the complexity. The packet complexity probe described in the previous section uses an entropy calculation technique for estimation of complexity. The Kolmogorov Complexity estimator, currently implemented as a simple compression estimation method, returns an estimate of the smallest compressed size of a string. The complexity *K(x)* is computed using the entropy *H(p)* of the weight of ones in a string. Specifically, *K(x)* is defined in Equation 1.A where is the number of 1 bits and is the number of 0 bits in the string whose complexity is to be determined. Entropy *H(p)* is defined in Equation 1.B. The expected complexity is asymptotically related to entropy as shown in Equation 1.C. See [10] for other measures of empirical entropy and their relationship to Kolmogorov complexity.

$$(\hat{K}(x) \approx l(x)H)\left(\frac{x\#1}{x\#1 + x\#0}\right) + \log_2(l(x)) \qquad \textbf{1.A}$$

$$H(p) = -p\log_2 p - (1.0 - p)\log_2 p - (1.0 - p) \qquad \textbf{1.B}$$

$$H(X) \approx \sum_{l(x) = n} P(X = x)K(X) \qquad \textbf{1.C}$$

The complexity estimation technique used here is not the best because empirical entropy is actually a very poor method of complexity estimation. For example, the estimate for the string

10101010101010101010

and a completely random string with equal numbers of 1's and 0's is the same under empirical entropy. More accurate estimates for complexity will only serve to improve our method for DDoS detection. See[162] for an innovative and improved method for complexity measurement. In future work, this technique will be used in the complexity probe and the performance of the algorithm will be compared with respected to the two techniques.

Experimental results

We compared our technique to a prototype packet counting algorithm for DDoS detection and found that our technique is better discriminates traffic patterns. We used our Magician-based [159] active network [160] test bed for the experiment for two reasons. It is quite easy to set up a desired topology for the network, as well as control and measure performance using an active network. Secondly, it is easier to embed our complexity probes, which are written in Java, inside the Java-based Magician kernel as opposed to embedding them inside commercial routers. The results, however, can be extrapolated to real traffic settings (Figure 54).

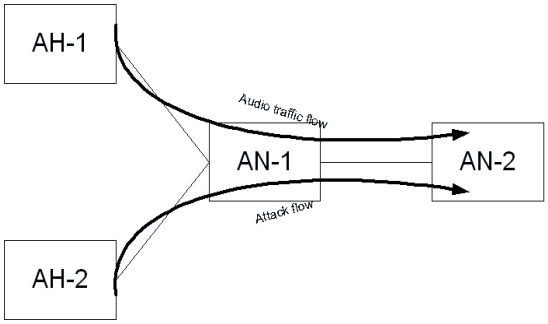

Figure 54. Topology for experiment.

The experimental setup consisted of a set of active nodes arranged in the topology shown in. Node AH-1 continuously generates traffic consisting of audio packets destined for node AN-2. The load induced by this traffic is high enough that it is registered at node AN-1 as a 'suspicious' flow i.e. a traffic flow whose complexity differential exceeds the threshold. The load induced by this traffic flow is kept constant throughout the experiment. Node AH-2 generates the attack flow. The load induced by the attack flow is varied to determine the performance of the algorithms. The experiment is run twice, once with only the attack source on (node AH-2 transmitting only) and the next time with both sources on (both node AH-1 and node AH-2 transmitting). The rationale is that an attack is essentially a sustained overload induced for some time interval.

The purpose of the experiment is to determine the effectiveness of the two techniques in separating and identifying an attack in the presence of background traffic.

Figure 55 and Figure 34 show the performance of the packet-counting and complexity-based approaches as measured against the load induced by the two sources (in packets per second) described above. Figure 55 shows that the packet-counting metric cannot discriminate between an attack and a true overload. When the audio source is transmitting in conjunction with the attack source, any threshold set by the packet-counting algorithm running on node AN-1 will be exceeded leading to the false conclusion that the node is under attack. For example, based on the attack pattern only (blue curve), we decide to set the threshold at 70 packets/s for a load of 0.6. When the audio source is introduced, the combined traffic trips the same threshold at a load of only 0.4, which is a false positive. Figure 54 below shows the complexity differential versus load curve for a given sampled time interval, which in this case was 10 seconds. The complexity-based metric does not change its behavior when a combination of attack and traffic sources is

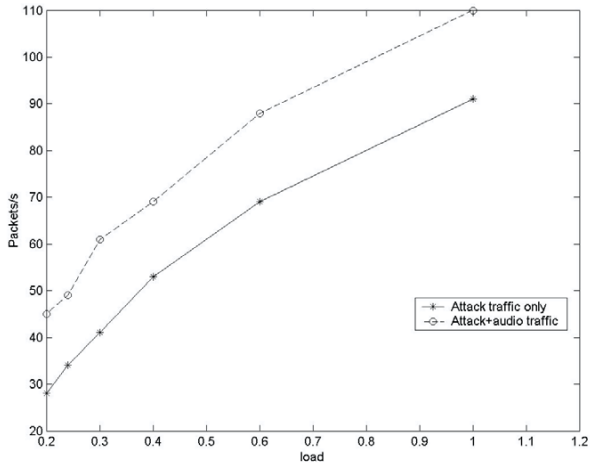

Figure 55. Performance of packet-counting metric.

used. This is because the attack traffic dominates the combined flow and hence the complexity differential roughly equal to that observed when only the attack flow existed. Therefore, the complexity-based approach is more accurate in separating false alarms from true attacks because it can conserve salient patterns of a traffic flow.

3. Kolmogorov Complexity as a Fundamental Metric Enabling Vulnerability Analysis

3.1 AUTOMATED DISCOVERY OF VULNERABILITIES WITHOUT A PRIORI KNOWLEDGE OF VULNERABILITY TYPES

The design of the vulnerability analysis tool consists of three logical layers as shown in Figure 56. Com-

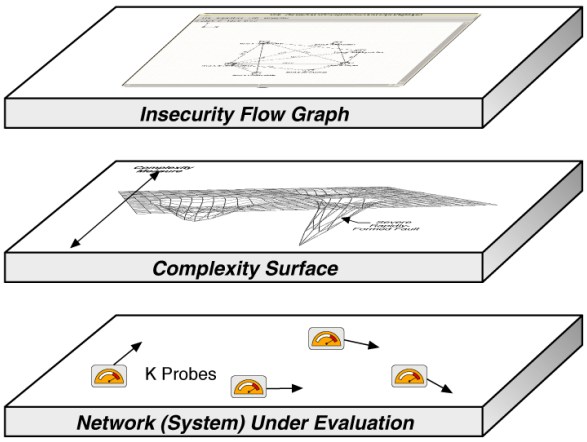

Figure 56. Logical view of complexity-based vulnerability analysis process.

plexity measurement probes within the actual system form the first layer. The results from the probes are used to build a K-Map. The K-Map consists of a $C \times C \rightarrow K|L$ matrix, where C is the set of information system components. The matrix represents a complete representation of "attack" components crossed with target components. The diagonal values are zero because the complexity of a component, assuming the component has already been compromised, is zero. Components that cannot physically be accessed from another component have an infinite complexity value. Note that the K-Map values change as an attack progresses. The complexity values are updated using conditional Kolmogorov Complexity estimates from Equation (1.2). This updates insecurity flow given that an attacker has partially penetrated the system and has gained knowledge of the compromised components. The result of this matrix can be viewed as a complexity surface as shown in [57]. The top layer consists of vulnerability states and transition values obtained from layer two. Relative complexity estimates are used to quantify the resistance to attack along the edges of the graph and the nodes are the state of an attack.

A model information system has been implemented in Mathematica [102]. Mathematica provides an ideal environment for experimenting with symbolic mathematical concepts and algorithmic information theory in general. The goal is to determine the vulnerability not only of the overall system, but also of system components. Vulnerability analysis must be possible without *a priori* knowledge about system operation or knowledge of particular types of vulnerabilities. Expert systems and vulnerability analysis tools that rely upon rules identifying particular types of vulnerabilities are inherently brittle. Such tools provide good performance when known attacks are applied, however, they fail catastrophically, and are therefore useless, against an innovative attacker.

Every information system is assumed to take data of some form as input, process the data and return data as output. Every information system can be defined as a mathematical operation. Information systems developed by humans today tend to be highly structured in order to be tractable in their development and maintenance. Generally, there are well-defined data flows and processing functions within the information system. The system is composed of a hierarchical composition of functional units. For these systems, one can imagine complexity probes located at the input and output of every functional unit in the system. This allows determination of the vulnerability of each process and data stream at a high degree of granularity. This provides a complexity-based vulnerability map for the system. A potential attacker would be unlikely to have such a detailed understanding of a target information system. An optimization to this technique is to limit probe locations to only those locations likely to be observable to an attacker.

System under evaluation: the active network

In the remainder of this paper, a specific example is used to communicate the architecture and operation of the vulnerability analysis framework. The spe-

cific example focuses upon an active network [102] in which a distinction is made between an active network and a legacy, or passive, network. This environment is used to emphasize that information assurance laws must be able to deal with many alternative and dynamically changing representations of information.

With regard to active packets and information theory, passive data is simple compressible data; active packets are a combination of data and program code whose efficiency can be estimated by means of Kolmogorov Complexity**.** A brief conceptual view of Kolmogorov Complexity for active network packet optimization is demonstrated in Figure 57 in which the same information is presented with varying proportions of code to data. The length of the information varies with the hypotheses used to represent the information within the packet. The shortest possible representation is the estimate of the packet complexity [109]. The active network Kolmogorov Complexity estimator is currently implemented as a quick and simple compression estimation method. The Kolmogorov Complexity estimator returns an estimate of the smallest compressed size of a string. It is based upon computing the entropy of the weight of one bits in a binary string. Specifically it is defined in Equation 6 where *x*#1 is the number of 1 bits and *x*#0 is the number of 0 bits in the string whose complexity is to be determined. Entropy is defined in Equation 7. See [103] for other measures of empirical entropy and their relationship to Kolmogorov Complexity. The expected complexity is asymptotically related to entropy as shown in Equation 8.

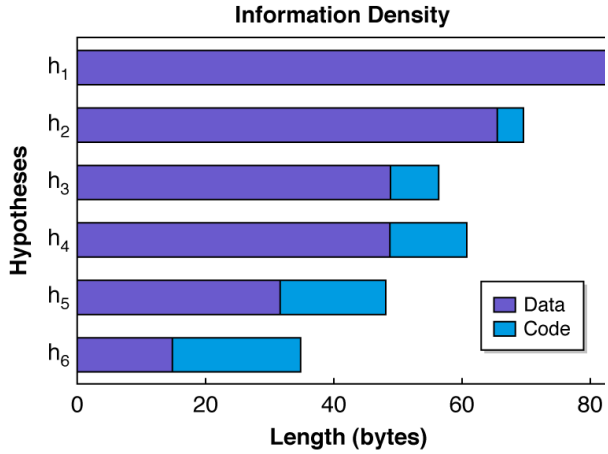

Figure 57. Same active packet information; varying hypotheses (proportion of code to data).

Observe an input sequence at the bit-level and concatenate with an output sequence at the bit-level. This input/output concatenation is observed for either the entire system or for components of the system. Low complexity input/output observations quantify the ease of understanding by a potential attacker. Previous work has demonstrated the use of Kolmogorov Complexity for Distributed Denial of Service (DDoS) attack detection [104]. Definition 6 explicitly states the means of measuring the complexity of a system component, or protocol interaction, to a potential attacker.

$$(\hat{K}(x) \approx l(x)H)\left(\frac{x\#1}{x\#1 + x\#0}\right) + \log_2(l(x)) \quad (6)$$

$$H(p) = -p\log_2 p - (1.0-p)\log_2 p - (1.0-p) \quad (7)$$

$$H(X) \approx \sum_{l(x)=n} P(X = x)K(X) \quad (8)$$

Definition 6: Complexity-based Vulnerability Metrics

*Vulnerability is inversely proportional to K(x[*Opstart:Opend*])/l(x[*Opstart:Opend*]) where* Opstart *is the bit at which an operation to be discovered within an information system begins, and* Opend *is the last bit in an attacker's observation*

In the remainder of the paper, excerpts from a Mathematica Notebook are included. The excerpts contain code using common mathematical and programming constructs, and therefore should be intuitively obvious without requiring knowledge specific to Mathematica. Any Mathematica specific details are explained in the text. As a specific example of the algorithmic capabilities of active networks, consider the transmission of an estimate of π. One could choose to send π as an extremely large number of digits. Or in contrast, one could send a smaller algorithm capable of generating π to an arbitrary number of digits. Consider an illustration of this concept in more detail. The Mathematica code, {{#1/#2 &}, {22.,7.}}, represents an unnamed function that divides the first argument by the second argument; the function implements 22/7. Consider that the code ({#1/#2 &}) and the data ({22.,7.}) remain unevaluated and are transmitted together. This represents an active packet; it contains part code and part data. The RUN function evaluates the function and returns the result. The result in this case is static data, a legacy data packet. Mathematica code that

analyzes the characteristics of algorithmic and passive information transmission is shown in Figure 58.

```
res = AnetSim[{{{#1/#2 &}, {22., 7.}},
    {{#1 &}, {RUN[{{#1/#2 &}, {22., 7.}}]}}},
   {{1, 2, 3, 4}, {4, 3, 2, 1}}, {100, 100, 1000, 10000}];
```

Figure 58. Static versus active information in the Mathematica active network simulator.

The active packet is defined as {{#1/#2 &}, {22.,7.}}, which contains a pair of values and the code necessary to perform division. The legacy, or passive packet, is defined as RUN{{#1/#2 &}, {22.,7.}}, which pre-computes the result of the division and transmits the same information in non-algorithmic form. The argument defined as {{1,2,3,4},{4,3,2,1}} identifies the links traversed by the active and passive packets respectively. In this case, the first packet begins by crossing link one and the second packet begins by crossing link four. The argument defined as {100,100,1000,1000} indicates link capacities for links one, two, three and four. Thus, the first packet transmits both code and data that generates the intended information, while the second packet transmits raw data only. The result of executing the function below is load and processing time spent on each link and node for each packet. In Figure 59,

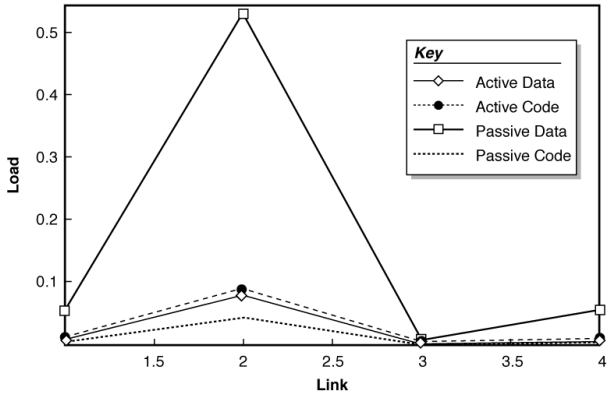

Figure 59. Algorithmic versus static active network information load.

the load induced by sending the estimate of π using AnetSim in Figure 58 is plotted for each link. Clearly, the algorithmic representation of the information is more compact and uses less link capacity.

In fact, this reinforces the fact that by knowing how to compute π, one could build a more compact representation. This demonstrates Occam's Razor for a useful purpose, information compression. This has facilitated study of active (algorithmic) versus passive transmission of information. For example, we allow the ratio of data to code to change for the same information as the packet traverses the network in a manner that optimizes both link capacity and node processor speed.

Complexity surface: the Kolmogorov Complexity map

The GE Global Research active network test bed implements complexity probes as part of the active execution environment. The choice was made to embed the complexity probe in the execution environment rather than as an active application because it is necessary to examine the content of active packets before they reach the execution environment. In the Mathematica simulation, each component of the active application contains probe-input points through which bit level input and output is collected. A complexity estimator based upon the simple inverse compression ratio from Equation (1.4) is used to estimate complexity in the density metric. Figure 60 and Figure 61 graphs result from

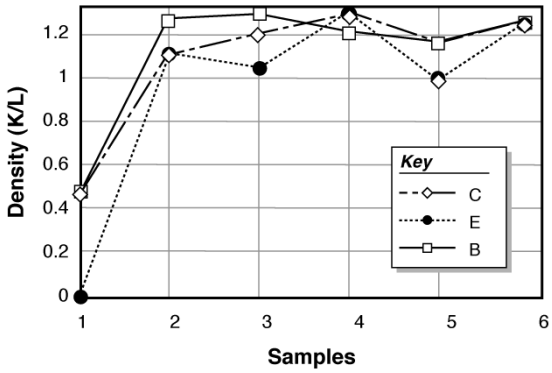

Figure 60. Component complexity for components B, C and E.

density estimates taken of accumulated input and output of three separate components of the active network application. The graphs show the complexity of bit-level input and output strings concatenated together. That is, every input sequence is concatenated with an output sequence and the density of the sequence is recorded at bit-level.

The input/output concatenation is generated either for individual components of the system or for a composition of components. If there is low complexity in the input/output observation pairs, then it is likely to be easy for an attacker to understand the system. The X-axis is the number of input

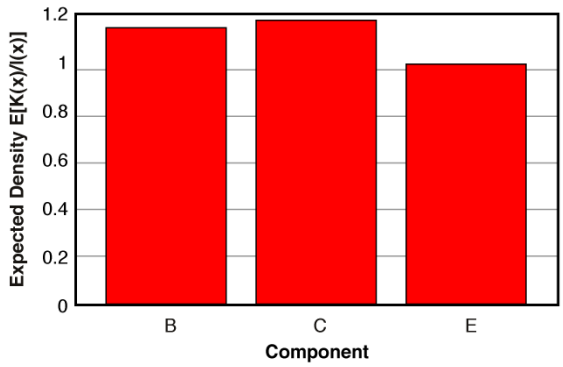

Figure 61. Mean component complexities for B, C and E.

and output obser vations concatenated to form a single string of bits. From Figure 61, it would appear that Component E is most vulnerable due to its consistently low complexity while Component B appears to be the least vulnerable due to its larger complexity. These results make intuitive sense because Component E simply forwards data without any form of protection while Component B adds noise to the data. This vulnerability method does not take into account whether a component reduces or increases complexity. In other words, whether the change was endothermic or exothermic complexity. These results demonstrate how vulnerabilities are systematically discovered using complexity. Vulnerabilities can be quantified to a value within the bounds of the complexity measure error.

In order to develop the Kolmogorov Complexity Map (K-Map), consider the topology in more detail. Figure 62 shows the resulting densities inserted into

```
gnp = Graph[KMap, Range[Length[KMap]]]
Graph[{{∞, 1.17693, ∞, 1.00975}, {∞, ∞, 1.1074,
  {∞, ∞, ∞, ∞}, {∞, ∞, 1.1074, ∞}}, {1, 2, 3, 4}]
```

Figure 62. Kolmogorov Complexity map (K-Map).

a Mathematica graph object. The graph object allows graph theory related analyses to be applied. The directed graph Figure 63 shows the relationship among the vulnerabilities. The START state, located in the center of the topology, represents a location outside the system. In Figure 64 a matrix is generated that shows the cost, in terms of complexity, of traveling from any node to any other node in the K-Map. In Figure 65, the function *CoordVul* computes a maximum flow through the K-Map graph using the

```
p3 = ShowLabeledGraph[g, {START, B, C, E}];
```

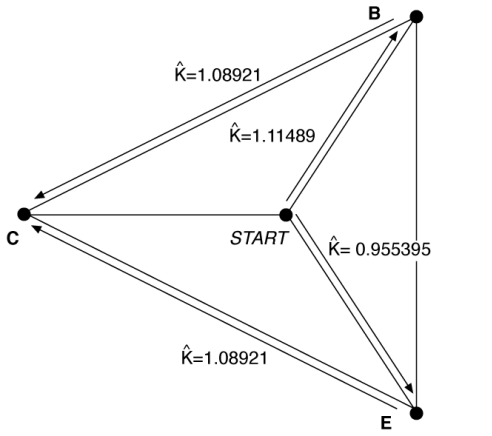

Figure 63. System under analysis: components and topology.

```
MatrixForm[AllPairsShortestPath[g]]
```

$$\begin{pmatrix} 0 & 1.11489 & 2.0446 & 0.955395 \\ \infty & 0 & 1.08921 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 1.08921 & 0 \end{pmatrix}$$

Figure 64. Minimum complexity paths matrix.

```
CoordVul[g_, x_, y_] := Module[{s},
        Return[ Sum_{s=1}^{Length[g[[2]]]} NetworkFlow[g, s,
    xy2node[g, x, y][[2]]] ]
]
```

Figure 65. Insecurity flow graph.

node positions as shown Figure 63. Density ($K(x)/l(x)$) acts as a resistance, while its inverse acts as conductance, supporting insecurity flows as illustrated in Figure 66. The resulting flow matrix in Figure 68 shows the maximum flow through each link. Figure 68shows the complexity surface of the resulting flows. Higher areas correspond to less vulnerable states, while lower areas correspond to more vulnerable states. Note that in the following contour maps, areas of infinite height are simply shown without a surface. By comparing Figure 63 and Figure 68, it is apparent that the START state, the infinite mountain in the center of the topology, is

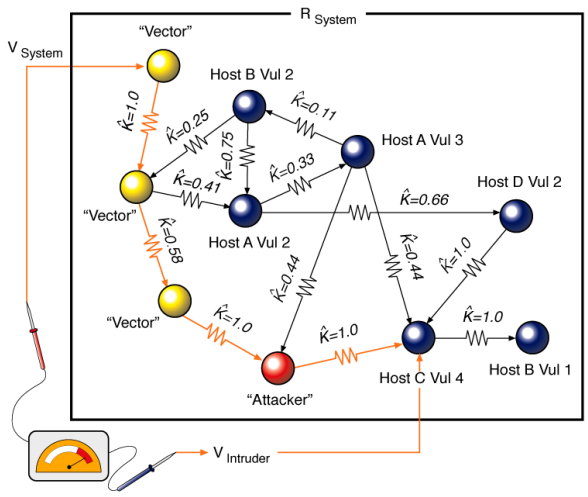

Figure 66. Grid-based representation of information assurance.

```
MatrixForm[Table[CoordVul[gnp, x, y],
  {x, -1., 1., .3}, {y, -1., 1., .3}]]
```

3.65124	3.65124	3.65124	3.65124	3.65124	3.65124	3.65124
3.65124	3.65124	3.65124	3.65124	3.65124	3.65124	3.65124
1.04669	1.04669	3.65124	3.65124	3.65124	3.65124	0.896949
1.04669	1.04669	0	0	0	0.896949	0.896949
1.04669	1.04669	1.04669	0	0	0.896949	0.896949
1.04669	1.04669	1.04669	1.04669	0.896949	0.896949	0.896949
1.04669	1.04669	1.04669	1.04669	0.896949	0.896949	0.896949

Figure 67. Flow results matrix.

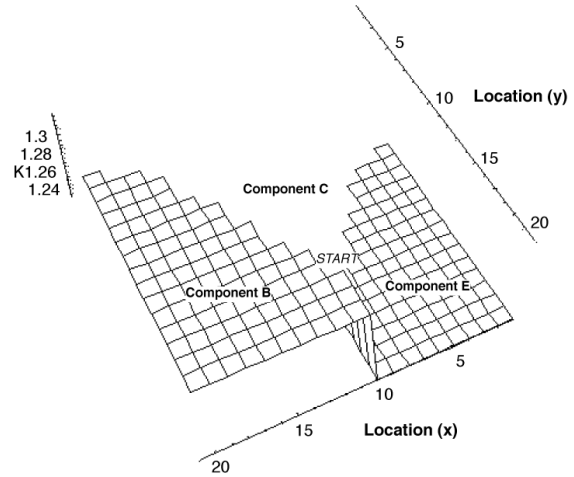

Figure 68. Complexity surface for system in Figure 84.

invulnerable, which makes intuitive sense. State E is the weakest individual component and lowest area on the right side. Note that while State C cannot be directly attacked from the START state, it can be attacked via states B and E, located in the upper and lower right side of the figure respectively. Thus, B and E have a relatively intermediate level of vulnerability. In the insecurity flow contour shown in Figure 69, density is resistance and all possible flows

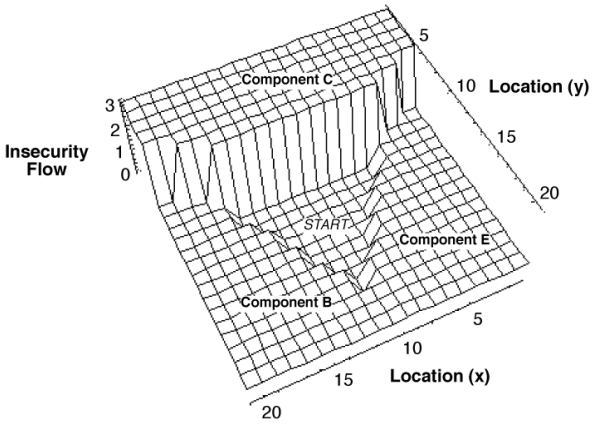

Figure 69. Insecurity flow contour of system in Figure 63.

from and to every node are summed to obtain an in security level. While Node C is assigned infinite complexity as shown in Figure 68, it actually is the most insecure component given that flows exist from Nodes B and E.

3.2 A PRIORI VULNERABILITY ANALYSIS: THE NETWORK INSECURITY PATH ANALYSIS TOOL

The Network Insecurity Path Analysis Tool (NIPAT) [105], like many security tools, assumed *a priori* knowledge of vulnerabilities. It then estimated security flow by assigning probabilities based upon the number of opportunities for an attacker to advance from one vulnerability to another. An example of NIPAT operation is shown in Figure 70. In this figure, 2,000 *a priori* defined vulnerabilities found on a few nodes of a network that were thought to be reasonably secure are displayed. The hosts upon which vulnerabilities reside and the *a priori* defined type of vulnerability are displayed. The number along each edge of the graph represents the number of opportunities available to the attacker to reach the next vulnerability. This information is gathered from network security software agents that are pre-programmed to identify predefined types of vulnerabilities. The security vulnerability graph for a typical network can be extremely dense, however, the object-oriented nature of the security model is

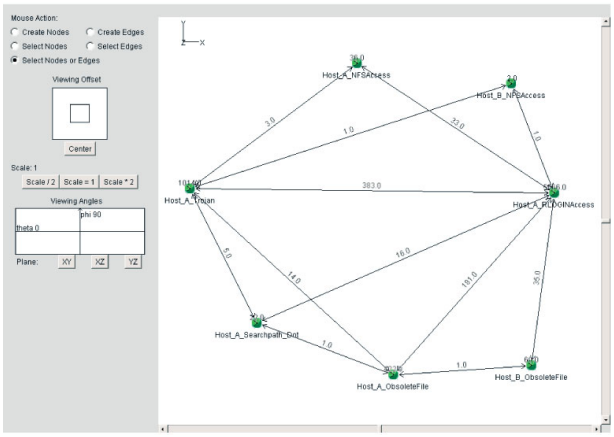

Figure 70. A grid-based tool action.

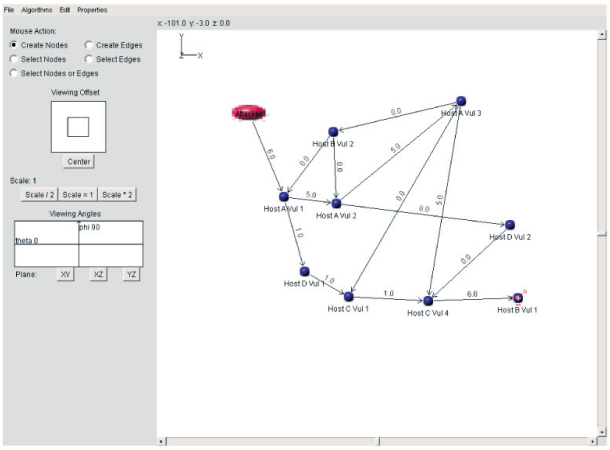

Figure 71. Most likely attack path.

useful in choosing the level of abstraction required. For example, it may be possible to display the vulnerability graph for Unix hosts in general and to hide the details of individual Unix variants. NIPAT determines the degree to which specified targets within the network can be compromised. The vulnerability chain is displayed as a directed graph. Nodes represent vulnerabilities whose security may be compromised, and edges represent paths from vulnerability to vulnerability. The larger the value of the edge label, the greater the vulnerability. The focus of this effort is on the mathematical representation of information assurance, thus, the underlying database and data gathering agents are not discussed in detail in this paper.

Two algorithms are demonstrated; the first is a probabilistic analysis and the second is a maximum flow analysis. Let us start with the probabilistic analysis. Select a node to be the target of the attack; in this case we have selected **Host C Vuln** 4. Select a specific attack entry point anywhere in the system and add the attacker to the graph. A text window appears stating the probability of successful attack (0.729) followed by the graph shown in Figure 71 that shows the most probable path of attack highlighted. The analysis is re-executed using the maximal flow algorithm. **Host C Vuln** 4 is again selected as the target. A text window appears displaying the maximum flow (6.0) as well as detailed graphical results shown in Figure 72. The edge values have been changed to show the maximum flow along each edge towards the target node. In this case there is a flow of 1.0 and a flow of 5.0 that can reach the target node.

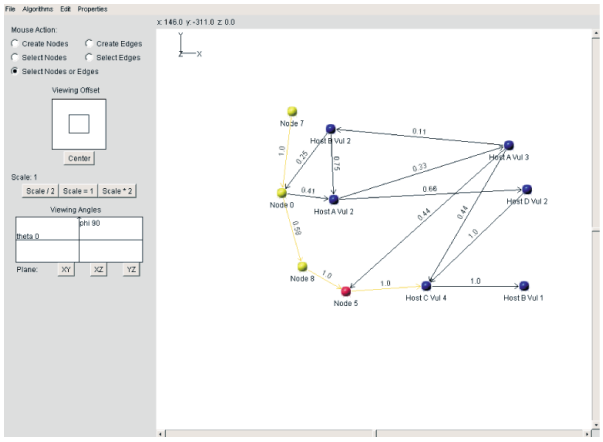

Figure 72. Maximum flow paths.

Complexity-based insecurity flow

Assigning probability of exploitation for vulnerabilities based upon the assumption that all vulnerabilities can be explicitly discovered *a priori* and placed in data or knowledge base is a fallacy for several reasons. First, a brute force approach that attempts to *a priori* explicitly identify all possible vulnerabilities is highly system dependent and results in a combinatorial explosion. Second, assigning a level of effort required to exploit vulnerabilities is highly subjective. Third, failure to identify even a single vulnerability can result in catastrophic performance failure. Such a brute force technique is very brittle as shown in the next section. Fourth, once such inaccurately quantified probabilities have been assigned, the probabilistic mechanism is an unsuitable technique. For example, the simple assumption does not follow

that composing components serially results in a probability of successful attack quantified by the product of the probabilities. Mutual information between the components can result in a much higher probability of successful attack.

Using results from the K-Map described in the previous section, it is possible to address these problems by retrofitting NIPAT to use the complexity-based vulnerability framework. Both a most likely path and a maximum flow algorithm are applied in this experimental complexity-based vulnerability analysis tool. The most likely path is determined by finding the lowest complexity path from a given attack point to a given target point. The maximum flow algorithm assumes that lower complexity paths have a greater capacity.

The question arises as to what *flow* means in terms of complexity. First, the entire foundation of complexity-based vulnerability analysis rests upon the likelihood, or probability, of attack being successful upon the low complexity locations of an information system as per Definition 6. The complexity probe values are displayed as links in the complexity tool display shown in Figure 71. The values of the links are $l(n)/K(n)$ and these values are normalized to 1.0 for each node in order to obtain a probability of successful attack upon each link. The maximum flow algorithm provided by this tool indicates not only the vulnerability of each component, but also the optimal placement of resources by an attacker to maximize the likelihood of a successful attack.

Safeguard optimization techniques

We assume that vulnerability has been calculated by NIPAT to be either the maximum insecurity flow or probability of successful attack, where S represents security safeguards, $C(S)$ is the cost of security, and L is the cost constraint or some other hard resource limit. Next, we discuss the cost in terms of impact on users. Here it is strictly a financial cost or other resource constraint. Objective Function 1 shows how the optimal security safeguard allocations can be determined. $V(S)$ is $l(S)/K(S)$. Let CS represent the network service to customers, with a minimum accepted quality, Q. Let $V(S,A)$ be the vulnerability of the network to a particular attacker, A. Then Objective Function 2 shows the optimal network response given the current state of the attack.

It is possible to use NIPAT to study various strategies of both the defensive and offensive players in a network attack. Once an attack has been detected, the network command and control center can respond to the attack by repositioning security safeguards and by modifying services used by the attacker. However, cutting off services to the attacker also impacts legitimate network users A careful balance must be maintained between minimizing the threat from the attack and maximizing service to customers.

Objective Function 1: Vulnerability and Cost	$\min V(S)$ $s.t. C(S) < L$
Objective Function 2: Vulnerability and Cost while Maintaining QoS.	$\min V(S,A)$ $s.t. C(S) > Q,\ C(S) < L$

The distribution of insecurity information—Another dimension of vulnerability analysis involves detecting vulnerabilities that change over time. The network monitoring tool quantifies the vulnerability of a system in terms of percent of patches that fail to have the correct signature, percent of files which are accessible to others besides the owner and percent of passwords which can be guessed with a given password generation tool. Clearly, vulnerability checks such as these increase the security of the network. Both the type of information gathered and the frequency with which the information is updated quantify the effectiveness of a network monitoring strategy. If the information is not updated frequently enough, an attacker may have penetrated network security and left before network security is aware of the situation.

An estimate of the effectiveness of the monitoring system is based on a profile of network security attacks on the Internet and the following parameters: time to monitor patches, Trojan horses, passwords, and any other vulnerabilities. The average attack rate, based on Internet incident reports from an anonymous site for a six-year period, is five attacks per month. Additionally, the Defense Information Systems Agency has determined by experimental means [107] that only 0.7 percent of incidents are actually reported. Thus, for each path in the network security vulnerability chain, the cost to the attacker is the probability of being detected multiplied by the cost function that the additional monitoring provides.

The approach to measuring the complexity of a system results in determining the ease with which a potential attacker can understand the system. It does

not directly account for the fact that information about the target system can be obtained by a potential attacker in algorithmic form, that is, in the form of an attack tool. Such a tool does not require the attacker to understand its operation. The attack tool is like an active packet, or a parasite that depends upon its host for transportation. This is distinct from a virus, whose primary function is replication and transport. For example, an attacker may have little understanding of a particular system, yet the attacker may download an attack tool that enables a successful attack. Thus, the distribution of attack knowledge needs to be considered. Once an attack tool is in the hands of an attacker, the apparent complexity is greatly reduced. There is an interesting feedback mechanism here; data that can reduce the apparent complexity to a potential attacker needs to be kept secure by the defender. Once obtained by an attacker, a significant drop in apparent complexity occurs, potentially leading to further significant reduction in apparent complexity as more vulnerability information is obtained and disseminated to other attackers.

One might view the evolution of complexity in the following terms. An information system is built. Initially, an attacker discovers its least complex components. The attacker decides to automate his attack (active) and/or publish the mechanism to accomplish the attack (passive). This information is disseminated through the population. Meanwhile the information system defenders, usually after considerable delay, discover the attack mechanism and patch the hole. The population of attackers, building upon their knowledge, exploits the next least complex link from their view in the information. The defenders eventually close this hole.

The cycle continues *ad infinitum*. The cycle of attack and defense can be viewed through complexity as a cycle, or evolution of complexity. Low complexity portions of a system will eventually be learned and disseminated by an attacker. To account for this dissemination of low complexity information, defenders reinforce the low complexity areas with more complexity. The results of this project allow system developers to understand not only where the vulnerable portions of the system are located, but to engineer their systems in such a manner as to control the cycle. This process can be modeled as low complexity portions of an information system that evolve in complexity over time.

3.3 INTRODUCTION

The vulnerability analysis technique presented in this paper takes into account the innovation of an attacker attempting to compromise an information system. A metric for innovation is not new, William of Occam suggested a technique 700 years ago [94]. The salient point of Occam's Razor and complexity-based vulnerability analysis is that the better one understands a phenomenon, the more concisely the phenomenon can be described. This is the essence of the goal of science: develop theories that require a minimal amount of information to be fully described. Ideally, all the knowledge required to describe a phenomenon can be algorithmically contained in formulae, and formulae that are larger than necessary lack of a full understanding of the phenomenon. The ability of an attacker to understand, and thus successfully innovate a new attack against a system component, is directly related to the size of the minimal description of that component.

Consider an information system attacker as a scientist trying to learn more about his environment, that is, the target system. Parasitic computing [95] is a literal example of a scientist studying the operation of a communication network and utilizing its design to his advantage in an unintended manner. The attacker as scientist generates hypotheses and theorems. Theorems are the attacker's attempts to increase understanding of a system by assigning a cause to an event, rather than assuming all events are randomly generated. If theorem x, described in bits, is of length $l(x)$, then a theorem of length $l(m)$, where $l(m)$ is much less than $l(x)$, is not only much more compact, but also $2^{(lx)-(lm)}$ times more likely to be the actual cause than pure chance [94]. Thus, the more compactly a theorem can be stated, the more likely the attacker is to be able to determine the true underlying cause described by the theorem.

Motivation

Imagine a vulnerability identification process that consisted of the following: First, wait for an information system to be attacked. Then analyze the attack, assuming, of course, the system survives the attack, can still be trusted and the attack can even be detected. Finally, if the information system is still not compromised, add the attack information to one's knowledge base.

This technique would be unacceptable to most people, but it is essentially the vulnerability analysis technique used today. Information assurance, and

vulnerability analysis in particular, are difficult problems primarily because they involve the application of the scientific method by a defender to determine a means of evaluating and thwarting the scientific method applied by an attacker. This self-reference of scientific methods would seem to imply a non-halting cycle of hypothesis and experimental validation being applied by both offensive and defensive entities, each affecting the operation of the other. Information assurance depends upon the ability to discover the relationships governing this cycle and then quantifying and measuring the progress made by both an attacker and defender.

A metric and framework are required for quantifying information assurance in an environment of escalating knowledge and innovation. Progress in vulnerability analysis and information assurance research cannot proceed without fundamental metrics. The metrics should identify and quantify fundamental characteristics of information in order to guarantee assurance. A fundamental definition of vulnerability analysis is formulated in this paper based upon attacker and defender as reasoning entities, both capable of innovation. Truly innovative implementations of attack and defense lead to the evolution of complexity in an information system. Understanding the evolution of complexity in a system enables a better understanding of where to measure, and how to quantify, vulnerability. In turn, this leads towards a calculus of information complexity. The design and implementation of a complexity-based technique is presented as a vulnerability analysis tool for automated measurement of information assurance. The motivation for complexity-based vulnerability analysis comes from the fact that complexity is a fundamental property of information and can be universally applied.

Components of the analysis

The presentation and analysis of a Kolmogorov Complexity-based vulnerability analysis framework must accomplish several goals. As initially stated, the vulnerability analysis technique must demonstrate the ability to account for the innovation of an attacker. The presentation should also discuss the relationship to previously defined properties of security. The technique should be based upon fundamental properties of information, rather than suffer from the combinatorial explosion that occurs when explicitly examining all possible events generated by specific systems. The vulnerability results should make intuitive sense; vulnerability is reduced by increasing the apparent complexity of access to information from potential attackers while increasing vulnerability for less complicated, or in some sense shortest paths of access to information. In other words, low complexity implies high vulnerability and high complexity implies low vulnerability. The results should not only be intuitively clear, but should support the rigorous definition of a metric space.

Once this has been shown, a topological view of vulnerability can be demonstrated. This is demonstrated by means of a Kolmogorov Complexity Map (K-Map) in which low complexity paths, which are likely to be easy for an attacker to follow, are identified. The concept of a K-Map, or complexity grid, is shown in Figure 73 and the K-Map for a specific

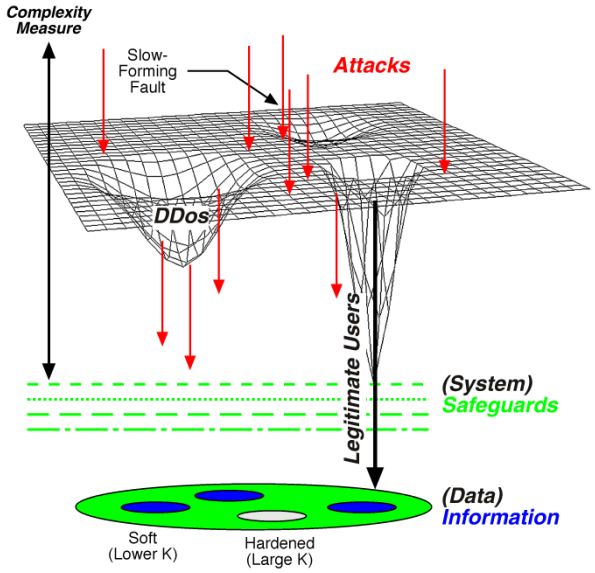

Figure 73. Conceptual view of a vulnerability and attack detection complexity grid.

example is derived later in this paper.

Figure 73 may itself appear quite complex upon first glance; however, focus upon individual parts of the figure in a logical progression. Begin with the information to be protected, which lies at the bottom of the figure. Attacks are illustrated as the thin, downward-pointing arrows attempting to penetrate the system in order to manipulate the information. Numerous safeguards, supposedly designed to protect the information and each designed to mitigate particular types of attack, are shown as barriers with various levels of porosity (inserted across the middle

of the figure). The overall complexity of the system is illustrated by the surface contour located above the information and safeguards. The complexity of the system as a whole is comprised of the complexity of several entities, namely: the information itself, the complexity of the system in which the information resides and the complexity of the safeguards. Innovative attacks will be more likely to successfully penetrate those areas of low complexity with easier to comprehend components of the system.

In addition, specific types of attacks, such as Distributed Denial of Service (DDoS) will appear as warps in the complexity grid. This is due the inherent system correlation in DDoS attack-streams. The vulnerability analysis technique should be applicable in a highly dynamic and amorphous information environment. An active network environment is chosen because information can be transmitted through an active network while its proportion of algorithmic content varies. In other words, static data, executable code or various combinations of both can represent information. In addition, both forms of information should have high assurance. The assurance of their interaction at a low level within an active network presents a nice challenge.

An example application of vulnerability analysis should be demonstrated to validate the feasibility of the framework. This paper ends by demonstrating several applications enabled by the new vulnerability analysis framework. The first application of vulnerability analysis shows that the complexity-based vulnerability framework enables Brittle Systems analysis. Brittle Systems analysis can be applied to understand the trade-off in performance versus failure of security. Finally, another application shows that complexity-based vulnerability analysis enables the optimization of security safeguards.

Properties of security

There have been many attempts to define security models that facilitate the proof of security properties [96]. The results in this paper focus upon what has been termed probabilistic, rather than possibilistic, security. Possibilistic security is concerned with proofs that given security properties can never be violated, while probabilistic security is concerned with estimating the likelihood that properties will be violated. The quantification of the insecurity that results from the successful exploitation of areas of weak security is referred to in this paper as vulnerability.

The security framework generally assumes that there are low-level and high-level users within a system. The intuitive notion is that high-level users should be secure from low-level users. Security properties include *non-inference*: low-level users should not be able to infer information about high-level users, *non-interference*: high-level users are prevented from influencing the behavior of low-level users (otherwise, low-level users could infer information about high-level user activity), *non-deducible output*: low-level users cannot distinguish the events causing high-level users' output, and finally *separability*: no interaction or information flow is allowed between low and high level users. Separability is too strong a security property because it does not allow low-level users to interfere with high-level users. This type of interference is acceptable, since it is assumed that information flow is allowed from low-level to high-level users. The *perfect security* property allows information to flow only from low to high-level users.

While in theory these properties are useful in attempting to prove that a system is secure, anecdotal evidence suggests that few developers will expend the effort required to ensure that their systems meet these properties. The number of events that must be verified for possibilistic security results in a combinatorial explosion. In contrast, this work attempts to develop a quantification of the degree to which a system has achieved perfect security using fundamental properties of information, rather than proving perfect security. Security properties such as non-inference, non-interference, non-deducible output, and separability, define various mechanisms by which information flow, that is, information that could be inferred by one class of user about another class of user, is prevented.

Similarly to previous work in this area, results in this work are based upon information flow generated by a low-level user, referred to as an attacker, inferring information about higher-level users. It is assumed that security is not discrete, but varies throughout a system and that attackers will want to follow paths of least resistance to obtain their objective. That is, an attacker will choose paths of least resistance with the possible constraint of optimizing for stealth or speed of attack. Probabilistic security has been explored in the past, however, obtaining values for probabilities of insecurity has generally been ill defined. This paper uses Kolmogorov Complexity [97] as an underlying means to estimate insecurity probabilities.

3.4 VULNERABILITY METRICS WITH PHYSICAL ANALOGS

Vulnerability is generally defined as the probability of a successful attack multiplied by the damage done by the attack. The paper focuses on predicting the probability of successful attack against an information system using fundamental properties of information. Information properties with physical analogs are explored because (1) they are likely to yield laws of information that are fundamental to information, not specific to individual systems, (2) the properties provide deeper insight to information assurance, and (3) they can be universally applied. Information properties that have physical analogs and that are candidates for fundamental parameters upon which to build information assurance techniques are briefly discussed in this section.

Volume

In his ground breaking 1949 paper, Shannon introduced fundamental trade-offs and limitations on the ability to transmit information across a channel disturbed by Additive White Gaussian noise (AWGN)[98]. This launched the science of information theory that has transformed the study of communications and coding of information. It also prompted the use of the term "bit" of information, which Shannon credits to J.W. Tuckey, into the mainstream literature.

The idea that information can be quantitized into bits (or sequences of yes or no answers to questions) is now well accepted, and one measure of the size of information is the number of bits used to convey the information. Information compression coding - both loss-less and lossy - as well as forward error correction coding, alters the size of the information in terms of bits by removing or adding redundancy. However, the unit of size, bits, is the term used to discuss the size of information, whether it is efficiently coded, error prone or self-correcting. Thus, while it is possible for information to change size without altering content, size is a fundamental property of information.

Entropy

Shannon entropy [98], also a fundamental property of information, measures the uncertainty of a random variable X based on the probabilities of each outcome. The entropy of a distribution defines the average per symbol compression bound in bits per symbol using a prefix free code. Entropy is derived from a given source distribution p of I symbols as shown in Equation (9). Kolmogorov Complexity, to be discussed in detail later, is estimated from an individual sequence of information. These two parameters are extremely powerful properties of information that occur at a fundamental level.

$$H(X) = -\sum_{i=1}^{l} p_i \log_2(p_i) \qquad (9)$$

Density, mass and energy

Density and mass, and their relation to energy, are properties of matter that have parallel, and intuitively pleasing, meanings in the information domain. Much research has taken place on the minimal energy required by an attacker to mount a successful attack. Density, like Kolmogorov Complexity, may measure the ability of a sequence to be compressed. Mass may simply represent the number of ones in a sequence, and energy, as in thermodynamics, may tie together quantities such as mass, density or entropy. The goal is to find parameters that can be observed directly from information sequences and compare objective quantities on which to base the science of information assurance. In the analytical framework developed in this paper, Kolmogorov Complexity is analogous to mass that is used to formulate a density metric.

Complexity

A contribution of the research presented in this paper is to utilize complexity, Kolmogorov Complexity in particular, as a fundamental property of information for vulnerability analysis. The definition of Kolmogorov Complexity rests upon the notion of a Turing Machine program. The Turing Machine is one of the most fundamental, general purpose computing abstractions and is well known in computer science. The Turing Machine consists of a seven-tuple $(Q, T, I, \delta, b, q_0, q_f)$. Q is a set of states, T is a set of tape symbols, I is a set of input symbols, b is a blank, q_0 is the initial state, q_f is the final state, d is the next move function, d maps a subset of $Q \times T^k$ to $Q \times (T \times \{L, R, S\})^k$. L, R, and S indicate movement of the tape to the left, right, or remaining stationary, respectively. There can be multiple tapes. Thus d implements a "next move" function. Given a current state and tape symbol, d specifies the next state, the new symbol to be written on the tape and the direction to move the tape. One approach to the study of security is to consider the Turing Machine program as a representation of normal system operation. In

such an approach, if the T uring Machine program recognizes, or accepts, an input string, then a user has gained access to the system. If the Turing Machine program accepts a string that we did not anticipate (Si), then the system is vulnerable, as stated in Definition 1. Clearly, the Turing Machine program is an abstract representation of any protocol implementation, or operating component operation. The set of unanticipated input strings that is accepted is the vulnerability of the component (*V*) as shown in Lemma 1.

Definition 1: System Vulnerability — *If a Turing Machine program recognizes, or accepts, a string, then the user entering that string is defined to have gained access to the system. If the Turing Machine program accepts a string that was not anticipated in the initial design of a system (S_i), then the system is vulnerable.*

Lemma 1: Secure Component — *Given V= S_i, a component is secure if and only if V=Ø.*

Throughout this paper the assumption is made that an attacker has the objective of exploiting any vulnerability that requires the attacker to understand enough of the component to design a system attack. The attacker must determine the function performed by the Turing Machine program. In this case, the attacker is assumed to have the ability to observe every member of the input language in order to deduce operation by viewing the output. The attacker is actually inferring d. As d is inferred, more opportunities for attack may present themselves.

Turing Machines and Kolmogorov Complexity

Information must be accessible to legitimate users while access is denied to potential attackers. This is accomplished by increasing the apparent complexity of access to information while providing legitimate users with enough *a priori* knowledge to reduce the apparent complexity. This leads one to conclude that complexity itself is a useful metric. However, the search for an absolute measure of complexity is a problem that may be equally as difficult as quantifying information assurance. There is a good reason for this; they are, in a sense, one and the same. The results in this paper demonstrate how complexity can be estimated for use as a system-wide vulnerability metric.

Kolmogorov Complexity is a measure of descriptive complexity that refers to the minimum length of a program such that a universal computer can generate a specific sequence. Kolmogorov Complexity is described in Equation 10, where *j* represents a universal computer, *p* represents a program, and *x* represents a string. Universal computers can be equated through programs of constant length; thus a mapping can be made between universal computers of different types. The string *x* may be either data or the description of a process in an actual system. Unless otherwise specified, consider *x* to be the program for a Turing Machine described in Definition 1.

$$K_{\varphi}(x) = \left\{ \min_{\varphi(p) = x} l(p) \right\} \tag{10}$$

$$K_{\varphi}(x|y) = \left\{ \begin{array}{l} \min\limits_{\varphi(p,x) = y} l(p) \\ \infty, \text{ if there is no } p \text{ such that } \varphi(p,x) = y \end{array} \right\} \tag{11}$$

Conditional Complexity, in Equation 11, quantifies the complexity of string *x*, given string *y*. Intuitively, it is the additional complexity of string *x* beyond that in string *y*. Conditional Complexity is used in developing the K-Map. A fundamental metric, based upon Kolmogorov Complexity, used throughout the remainder of this paper is density. Density and its inverse, dispersion, are shown in Definition 2. If *x* represents a program, then dispersion can be considered inefficiency in implementation in terms of size. A disperse implementation of a system has more transitions and states than necessary. Thus, there is greater opportunity for an attacker to find a weak point in the system. However, once an attacker breaks into a disperse system, there will be, on average, more energy, that is, longer string length, required to reach the attacker's target. Greater dispersion should imply reduced brittleness (Definition 8) of resistance to attack.

Definition 2: Density — *The density of x is K(x)/l(x), where l(x) is the length of x. Dispersion is the inverse of density.*

Complexity as a Vulnerability Metric

Information assurance is increased by increasing the apparent complexity of access to information from potential attackers while providing legitimate users the least complex, or in some sense the shortest

path, to access of infor mation. Figure 74 conceptu-

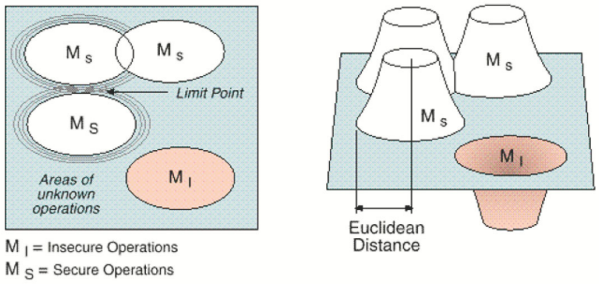

Figure 74. Set Theory View of Secure Operation.

ally illustrates an instance of secure and insecure operation in a system. Secure and insecure operation exist to varying degrees in the space of all possible forms of operation, *M*. Insecure operation, M_I, consists of those methods of operation that allow an information warfare aggressor entrance into, or access to control points, of the information system. The intended secure operation areas M_S are well known and some of the insecure paths are also known. Note that M_S and M_I can, and usually do, overlap. However, the entire area of operation can be extremely large and an exhaustive search for all insecure operation is not feasible.

In Figure 74, Euclidean distance corresponds to the degree of security. This leads one to consider a metric space upon which to base information assurance. The initial approach assumes only that the metric has the characteristics of a metric in the mathematical sense as shown in Definition 3, where *d* is distance and *p* and *q* are points.

Definition 3: Properties of an Information Assurance Metric	**(A)** $d(p,q) > 0$ *if* $p \neq q$; $d(p,p) = 0$ **(B)** $d(p,q) = d(p,q)$ **(C)** $d(p,q) \leq d(p,r) + d(r,q)$ *for any* $r \in X$

As illustrated in the left side of Figure 74, an information system de-composed into many operating components could have a surface area as shown on the right side of Figure 74. Note that this surface is likely to change as a function of time, however, the time indices are not written for now. The points, *p* and *q*, are assumed to be relative to some absolute value; *p* and *q* can be security values in either different locations or at different time instances of the system. If *d* is a measure of security, then Definition 3.A implies that there is no difference in security between the same point and itself. However, there must be a difference between any two distinct points in the security space. Definition 3.B states that the measure between any two points in this space should be the same regardless of the order in which one takes the measurement. This means that, observed from a common viewpoint, if security is measured at two different points in this space, *p* and *q*, then the measure of security will be the same regardless of the order in which the points are entered in the measure.

It does not imply anything about the strength of an attack from *p* to *q* or an attack from *q* to *p*. It means, for example, that if *p* is less than *q*, then an attack from outside the system against *p* will be more likely to succeed than an attack against *q*. Finally, Definition 3.C states that the distance between any two points will be less than or equal to the sum of the distances between each of those points and a common third point. Again, remember that this is a measure of security taken from a view outside the system of a potential attack from outside the system.

As discussed in more detail in the remainder of this report, the actual measure will change as an attacker penetrates the system and gains more knowledge of the system. Kolmogorov Complexity has been shown to possess the characteristics of a metric space [99] and in Section III an implementation is developed that generates a topology similar to Figure 74 for a given system.

If information assurance can be proven to reside in a metric space, or alternatively, if a metric space can be chosen in which information assurance can reside, then principles of mathematical analysis [100] can be used to rigorously determine more detailed characteristics. For example, *M* can be extremely large, possibly infinite. Are M_S, or conversely, M_I, open sets? If so, can limit points be defined? What does an open set mean with regards to information assurance and security?

As a simple example, consider a password protection system. Each character that a legitimate user of the system adds to a password increases the number of possibilities that a brute force (non-dictionary) attack would require in order to guess the password. Thus, the longer the passwords or encryption keys, the more secure the system. While an infinite length password is not possible, security does begin to approach a limit point.

This can also be seen in any security safeguard that works via the increase of complexity. That is,

adding more non-redundant states to a T uring Machine program, given the definition of performance in Definition 7, to increase security. This approach towards safeguard design approaches a limit point but can never reach perfect security. Security performance becomes less dense and less brittle. However, in general, this appears to be the only known approach, and thus limit points must exist.

Topological Space for Information Assurance

By definition, an open set (E) is one in which every point is an interior point. A point, p, is an interior point of E if there is a neighborhood, N, of p such that $N \subset E$. A neighborhood $N_r(p)$ *of* point p consists of all points q such that $N(p,q) < r$ where r is called the radius of the neighborhood. If security, as determined by a given metric, is an open set, then there are significant implications because of this. The best that can be hoped for in such a case is to determine limit points, because a distinct boundary between security and insecurity would not exist. Will it be the case that adding layers of security is much like adding "open covers", that is, the result can never be perfect security, but rather an approach to a limit point? The complement of an open set is closed; what does that imply for assessment of insecurity? Vulnerability analysis tools have been developed that assume all vulnerabilities have been identified and measured, and that the vulnerabilities can be manipulated as discrete, closed sets.

In order to determine whether such measurements can be applied to information assurance, consider topology, metric spaces, and the fundamentals of measurement theory in more detail. The definition below shows how the topology is induced by a metric d. In Definition 4, τ is a collection of subsets of X such that $\varnothing \in \tau$ and $X \in \tau$, any finite intersection of members of τ is in, and any union of members of τ is in τ. Definition 5 illustrates the topology that will be induced. This is particularly important in the development of the K-Map discussed in detail in Section III. In the vulnerability framework presented in this paper, the metric (d) is density (K/L) in Definition 6.

The intuitive notion is that d represents the ease of movement of an intruder from one vulnerability to another vulnerability, where $d(x,y): X \times X \rightarrow R$. A simple metric, as discussed previously, is to define d as the number of state/transition sequences, within a Turing Machine program representation of a system, which an intruder can follow to move from vulnerability x to vulnerability y or equivalently the cardinality of the set of V from Lemma 3. Does information assurance reside within this metric space? One test would be whether the metric supports the design tradeoffs required in determining brittleness in the design of the system.

Table 6

Definition 4: Metric Space	*Let d be a metric on X. A metric space (X, d) is a topological space where the topology Ù is the smallest one that contains all sets of the form {y: d(x, y)<Â} for all x and Â.*
Definition 5: Induced Topology	*V is the set of vulnerabilities and (V,d) is the topology induced by the choice of information assurance metric.*

To answer the above question, let V be the set of currently exploited vulnerabilities. Most information security approaches, including the one above, assume that all vulnerabilities have been discovered and measured. This can never be assumed to be the case. Performance (α) from Definition 7 is an open set, and as new security holes are discovered, $\lim_{|V| \rightarrow \infty} \alpha = 0$. If V represents vulnerability and is open, then secure operation, $\bar{V}$, is closed. Assume that $\sup_{x \in \bar{V}} d(x, x_0) < \infty$ for any x_0. Note that x is now an element of the set of secure operations. In other words, the number of secure operations is bounded. It is well known that a set is compact if and only if it is closed and bounded.

Building upon Definition 4 and Definition 5 requires that Turing Machine program states (Q) be identified as either secure or insecure. If an attacker can reach a member of $q_{Insecure}$ then the attacker is considered to have performed a successful attack. If an attacker can never reach a member of $q_{Insecure}$ then the system is considered invulnerable. The challenge is that neither the attacker nor the defender knows the entire structure of the Turing Machine program, first because the attacker is unlikely to have complete knowledge of the defender's system and also because even the defender may not fully understand the system that was developed. However, complexity estimation can be applied without requiring a detailed understanding of the target system. The following section presents results on the feasibility of the complexity-based vulnerability analysis technique by applying it to an active network.

3.5 BRITTLE SYSTEMS, DETERMINISTIC FINITE AUTOMATA, AND VULNERABILITIES

In order to understand the requirements for a vulnerability analysis metric; consider the manner in which systems that implement information assurance can be designed using such quantification. Design involves the tradeoff of one benefit for another. Brittle Systems Theory provides a framework for understanding the tradeoffs in performance versus failure of information systems.

Brittle systems analysis [108] is based on the idea that systems can fail in a manner analogous to brittle fracture of materials. A system can maintain very high performance until it fails quickly and catastrophically, as illustrated by performance curve Ph in Figure 75, or systems may fail by exhibiting lower

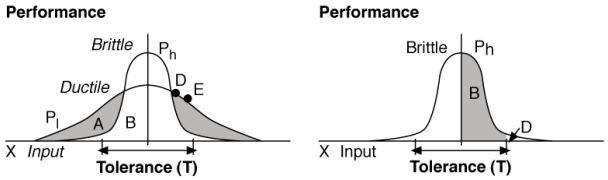

Figure 75. Definition of brittleness: brittle versus ductile performance.

performance in a gradual, more ductile manner as in curve Pl. The mapping between Brittle Systems Theory and information assurance is shown in Table 1. This analysis can be directly applied to the Turing Machine program representation of a system. Changes in any of the state machine parameters, Q, T, I, δ, b, q_0, q_f, may modify the brittleness of the system. For example, addition of a new state and transition could cause the system to behave in a more ductile or brittle manner. What is the measure of performance in a Turing Machine program model of information assurance? What does catastrophic failure mean in a Turing Machine program model of information assurance?

In order to answer these questions, the performance a from Definition 7 is resistance to attack and X in Figure 75 is effort required by the attacker. The answers to the above questions are intimately linked to the choice of metric. Based on Definition 1, one could choose the metric to be the number of state / transition paths available to an attacker to reach a particular target state, or equivalently, the cardinality of the set of strings (V) given in Lemma 1. Another possible metric could be the proximity of the attacker's current state to the state that is the target of an attack. Note that later it is shown that K/L is a vulnerability measure and is related to $|V|$. Next, more detail on Brittle System analysis and how it relates to the complexity-based vulnerability metric is discussed using Finite Automata.

Definition 7: Information Assurance Performance. *Information assurance performance is the inverse of the vulnerability induced by the choice of metric, $\alpha = 1/|V|$. A nearly invulnerable system has nearly infinite performance and an extremely vulnerable system has nearly zero performance.*

A Deterministic Finite Automaton (DFA) consists of a 5-tuple (S, I, a, $s0$, F) where S is the set of states, I is the input alphabet, a is a mapping from S into I, $s0$ is the start state, and F is a subset of S called the final, or accepting states. A DFA is less powerful than a Turing Machine program. However, DFA have been well studied and facilitate a framework in which new theories related to information assurance can be studied. An example of Brittle Systems using Definition 1 for vulnerability is illustrated for the DFA shown in Figure 76. A single vulnerability is rep-

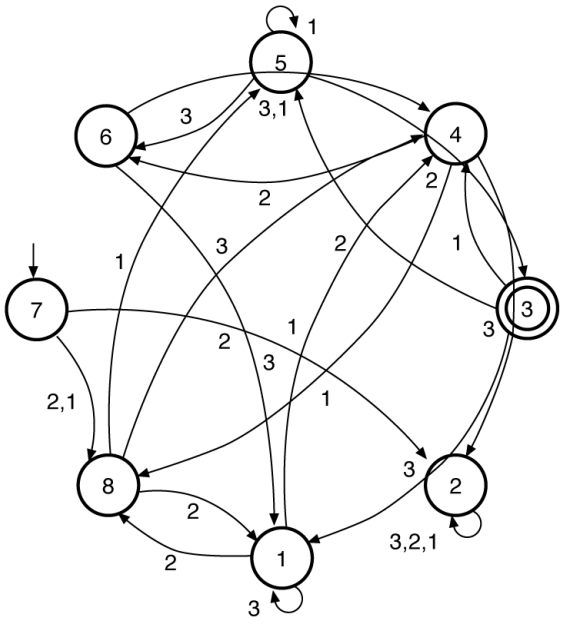

Figure 76. Example deterministic finite automaton of a system undergoing brittle analysis with the complexity-based vulnerability metric.

resented as a single modified transition, t_{faulty}. The modified transition represents an error in either the design or implementation that may allow an attacker to penetrate the system. The effect of each transition modified from its original source node to each possible destination node in the automaton is exhaustively checked. The effort expended by an attacker is assumed to be proportional to the length of the strings used in the attack. In the application of brittleness to vulnerability analysis, performance P is defined by a (Definition 7); X is defined by Als, which is the effort of an attacker measured in terms of string size required to reach an unintended accepting state. The algorithm requires starting with the actual system as represented in Figure 77, modi-

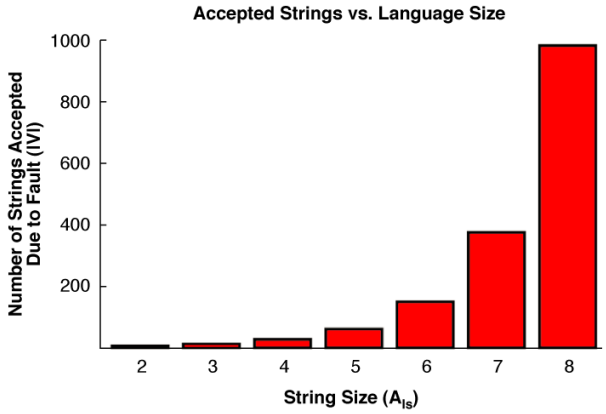

Figure 77. Ductile resistance to attack for system in Figure 79 with fault (7, 3, 2).

fying a transition and then recording the number of additional strings accepted. This is repeated for each transition in the base system. As shown in Figure 78, a modification of the transition from State 7, Input 3, Destination State 2, (7,3,2) yields a small number of vulnerabilities at string length two with a maximum of 1000 vulnerabilities at string length 8. This performance is ductile compared to the graph shown in Figure 78, where transition (1,3,1) is modified. Figure 78shows more brittle behavior because it takes a longer string length, thus more effort by the attacker to find vulnerabilities; however, the vulnerability increases rapidly as the string length increases. A more precise definition of Brittleness is given in Definition 8. Brittleness, defined by the area given in the definition, is in units of $Als/|V|$.

Definition 9 provides a means for easily computing complexity in the world of finite automata. Next the relationship between brittleness and complexity

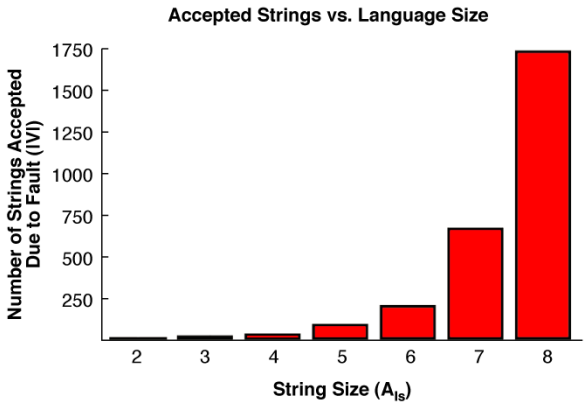

Figure 78. Brittle resistance to attack for system in Figure 79 with fault (1, 3, 1).

is addressed. One might intuit that a faulty transition in a less complex automaton will have less of an impact than a faulty transition in a complex version of the equivalent automaton. The definition of equivalent automata is given in Definition 10.

Definition 8: Brittleness.	*The brittleness of a system is a relative measure based upon the size of the area defined by* $\int_T (B - A)dT$ *.from Figure 79, where A and B are normalized to have the same area and T is a tolerance range. In the operation below, T is defined as the width of the line formed by the intersection of A and B*
Definition 9: Complexity of NDFA.	*he complexity of a DFA or NDFA is the number of transitions in the smallest DFA that accepts the original language of the DFA.*
Definition 10: Equivalence of Automata A and B.	*Automatons A and B are equivalent if and only if A and B accept the same language.*
Definition 11: Correlation between Brittleness and Complexity.	*There is a correlation between brittleness and density. A dense system, being more highly optimized, will fail at a faster rate than a simple version of the same system. On the other hand, a simple system will, on average, have more opportunity for error, while those errors are less catastrophic.*

Figure 79 and Figure 80 show a simple and complex implementation, respectively, of the same information system. Figure 79, as a simple implementation, is what might be intuitively referred to as an inefficient implementation,

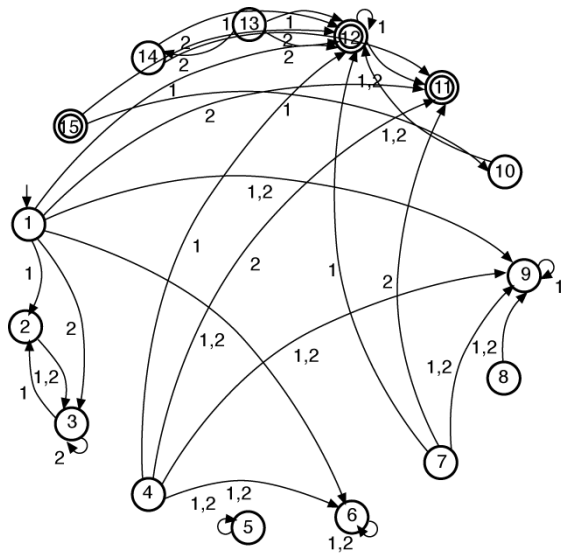

Figure 79. A disperse (low density) DFA.

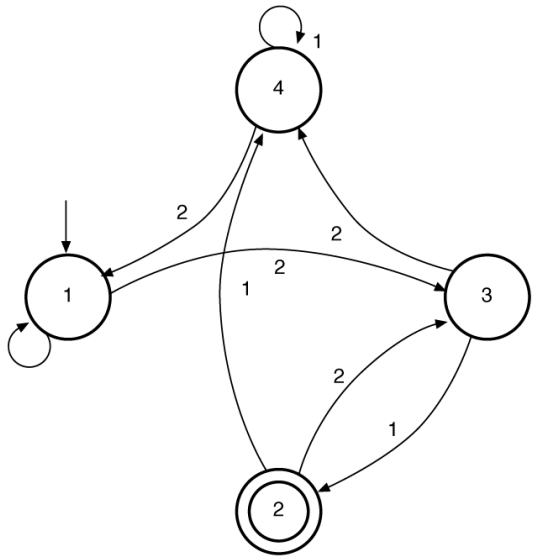

Figure 80. Implementation of the DFA in wFigure 84that approaches true complexity.

because it contains many more states than necessar y. This yields the opportunity for more vulnerabilities and faults. However, it also requires more effort by the attacker, that is, a larger $A_{ls,}$ to reach a given target.

Table 7 Brittle vulnerability analysis definitions .

Materials Science	Brittle Systems	Information Assurance
Stress	Amount parameter exceeds its tolerance	Applied force under the weight of an attack, A_{ls_T}
Toughness	System Robustness	Encryption strength and sensitivity of intrusion detectors
Ductility	Level of Performance outside Tolerance	Ability of system to gracefully degrade given an attack, $A_{ls}/\|V\|$: $A_{ls} > A_{ls_T}$
Plastic Strain	Degradation from which the system cannot recover	Trojan horse
Brittle Fracture	Sudden steep decline in performance	Sudden catastrophic collapse of all information assurance
Young's Modulus	Amount tolerance exceeded over degradation	
Deformation	Degradation in performance	The amount by which vulnerability has been increased due to an attack, ~ $(1/\|V\|)$
Brittleness	Ratio of hardness to ductility; estimated as difference in performance curves when outside tolerance	$(A_{ls_h}/\|V\|)$-$(A_{ls_d}/\|V\|)$: $(A_{ls_h} < A_{ls_T}$ and $A_{ls_d} > A_{ls_T})$
Ductile Fracture	Graceful degradation in performance	Ability of information to gracefully degrade under an attack
Reversible Strain	Degradation from which the system can recover	Trojan horse detection and removal

Table 7 Brittle vulnerability analysis definitions (Continued).

Materials Science	Brittle Systems	Information Assurance
Hardness	Level of Performance within tolerance limits	Resistance to decryption, A_{ls} /IVI: $A_{ls} < A_{ls_T}$

Figure 80 is a closer representation of the tr ue complexity of the same system. It has fewer opportunities for failure; however, the failures that occur will have a more significant impact. In Figure 81 and

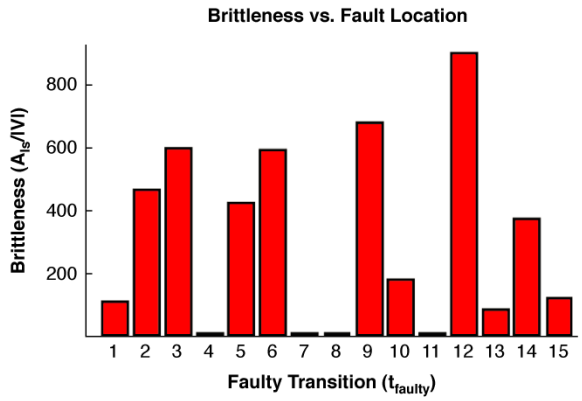

Figure 81. Brittle measure of DFA shown in Figure 78 in dimensions of A_{ls}/|V| versus t_{faulty}.

Figure 82, brittleness and complexity are compared.

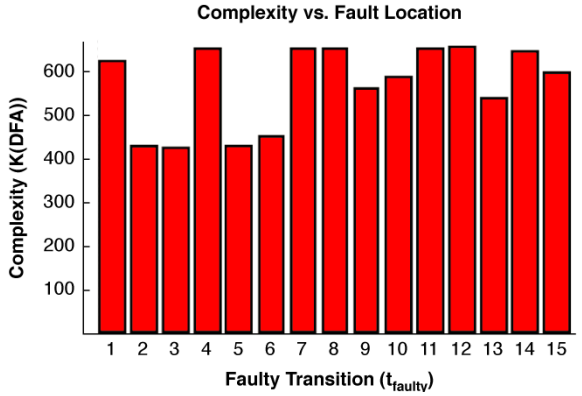

Figure 82. Complexity of DFA shown in Figure 78 in dimensions of *K(DFA)* versus t_{faulty}.

Brittleness is computed as defined in Definition 8. Performance is defined based upon the number of accepted strings and string size. The ratio of the number of accepted strings to total string size is inversely proportional to the performance. For each possible fault, this ratio is compared to a consistent base case consisting of an exponentially growing number of accepted words as the string size increases. A brittle system accepts few words initially, and then suddenly accepts a large number, while a ductile system accepts a moderate, but gradually increasing number with no sudden increase. The brittle measure is graphed as a function of a fault in the state specified on the dependent axis. A fault is generated by the re-connection of a single specified transition to a destination other than that which was originally specified. A single fault leads to many n-1 possible faulty states where n is the number of original states. Complexity is estimated as the number of transitions in the smallest representation of the resulting faulty system's DFA. Comparing Figure 81 and Figure 82, there appears to be an opposite relationship between brittleness and complexity. That is, a system with greater complexity results in lower brittleness. Greater complexity indicates a larger number of transitions and states exist, thus there is more opportunity for an attack, but more effort is required by the attacker to successfully complete the attack. In Figure 83 and Figure 84 a similar analysis

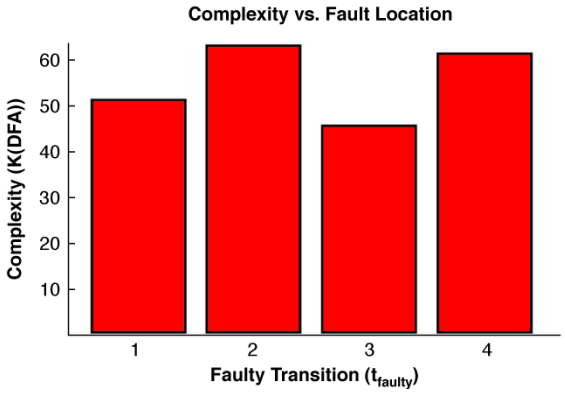

Figure 83. Complexity measure of system shown in Figure 79 in dimensions of *K(DFA)* versus t_{faulty}.

is performed on the more compact, or truer representation of the complexity, of the same system. A system with an implementation whose size is closer to its true complexity is more brittle. The inverse relationship between complexity and brittleness holds in the more compact system (Figure 80) as well.

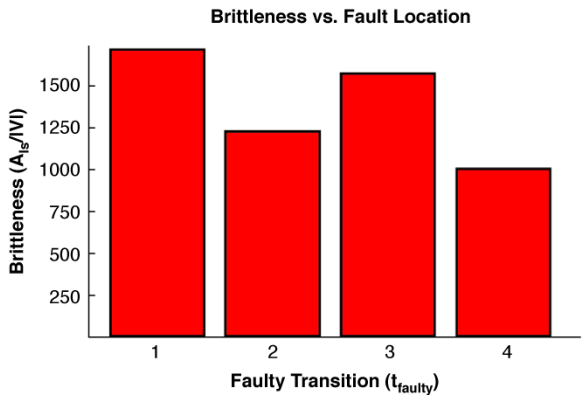

Figure 84. Brittle measure of system shown in Figure 79 in dimensions of $A_{ls}/|V|$ versus t_{faulty}.

An impor tant result in this exploration of the relationship among vulnerability, complexity, and brittleness is that the greater the dispersion, the lower the brittleness. This suggests that larger systems, requiring traversal of larger numbers of states and transitions to reach an accepting state, or target of attack, require more effort to successful attack. A system that has a large amount of inherent complexity cannot be designed more compactly than the length of its Kolmogorov Complexity. An intelligent attacker may be able to observe an inefficiently implemented system and reduce it to its most compact form, that is, its Kolmogorov Complexity, thus easily identifying paths of attack to reach specific targets. A truly safe system is thus obtained, not by building inefficiency into the system, but rather, by making the view to the attacker as inherently complex as possible.

Results of Work

4. Active Networks

Active Virtual Network Management Prediction enhancement via Kolmogorov complexity estimation

Kolmogorov Complexity ($K(x)$) (see [10] for an introduction to Kolmogorov Complexity and [5], [6], [8], and [9] for applications) is the optimal compression bound of string x. This incomputable, yet fundamental property of information has vast implications in a wide range of applications including network and system optimization, security, and Bioinformatics. Active networks [3] form an ideal environment in which to study the effects of tradeoffs in algorithmic and static information representation because an active packet consists of both code and static data. A question active network application developers must answer is, "What is the optimal proportion of packet content that should be code versus data?" A method for obtaining the answer to this question comes from direct application of Minimum Description Length (MDL) ([10] and [14]) to an active packet. Let D_x be a binary string representing x. Let H_x be a hypothesis, in algorithmic form, that attempts to explain how x is formed. MDL states that the sum of the length of the shortest encoding of a hypothesis about the model generating the string and the length of the shortest encoding of the string encoded by the hypothesis will estimate the Kolmogorov Complexity of string x, $K(x) \approx \mathrm{K}(H_x) + \mathrm{K}(D_x|H_x)$. A method for determining $K(x)$ separates randomness from non-randomness in x by incorporating non-randomness, which is computable, as the shortest encoded program that represents the original string. The random part of the string represents the error, that is, the difference between the original string and the output of the encoded program. Thus, the goal is to minimize $l(H_e) + l(D_x|H_e) + l(E)$ where $l(x)$ is the length of string x, H_e is the estimated hypothesis used to encode the string (D_x) and E is the error in the hypothesis, $D_x - (D_x \mid H_e)$. The more accurately the hypothesis describes string x, the shorter the encoding of the string. An active packet is measured as shown in Figure 85, where choosing an optimal proportion of code and data minimizes the packet length. The goal is to learn how to optimize the combination of communication and computation

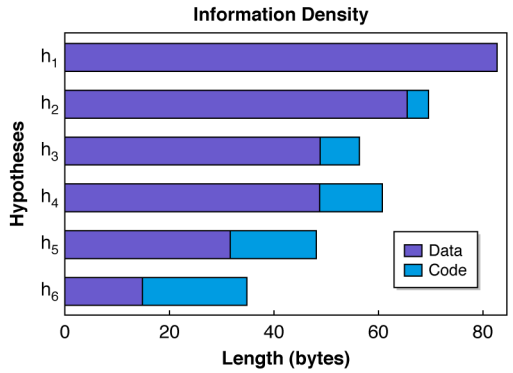

Figure 85. Algorithmic content.

enabled by an active network. Clearly, if is estimated to be high for the transfer of a piece of information, then the benefit of having code within an active packet is minimal. On the other hand, if the complexity estimate is low, then there is great potential benefit in including it in algorithmic form within the active packet. When this algorithmic information changes often and impacts low-level network devices, then active networking provides the best framework for implementing solutions (a specific example of separating non-randomness from randomness, although not explicitly stated as such, can be found in mobility management as discussed in [3] and [11]).

An active packet that has been reduced to the length of the best estimate of the Kolmogorov Complexity of the information it transmits will be called the minimum size active packet. When the minimum size active packet is executed to regenerate string x, the $D_x \mid H_e$ portion of the packet predicts x using static data () to correct for inaccuracy in the estimated hypothesis. There are interesting relationships between Kolmogorov Complexity, prediction, compression and the Active Virtual Network Management Prediction (AVNMP) mechanism described in [3]. These relationships are discussed and experimentally validated throughout this paper. The next section provides an overview of AVNMP before discussing its relationship to Kolmogorov

Complexity[2]. After required relevant background on AVNMP is explained, the relationship to Complexity Theory is developed beginning from a high level overview, then driving down into detailed relationships and experimental results.

4.1 ACTIVE VIRTUAL NETWORK MANAGEMENT PREDICTION OVERVIEW

The Active Virtual Network Management Prediction (AVNMP) architecture provides a network prediction service that utilizes the capability of Active Networks to easily inject fine-grained models into the communication network to enhance network performance. AVNMP, injected into the network as an active application, is capable of modeling load and propagating state information in a manner that meets the demand for accuracy at a particular active node. Greater demand for prediction accuracy is met at the cost of AVNMP performance, that is, the ability of AVNMP to predict farther into the future. While this paper focuses on network traffic and load prediction, an AVNMP application to predict CPU utilization for Active Networks in collaboration with National Institute of Standards and Technology ([4], [12] and [13]) has been demonstrated. The inherently distributed nature of communication networks and the computational power unleashed by the Active Networking paradigm have been used to mutual benefit in the development of the Active Virtual Network Management Prediction mechanism. Active Networks benefit from AVNMP by continuously receiving information about potential problems before they occur.

AVNMP benefits from Active Networks in many ways. The first, and most practical way, is the ease of development and deployment of this novel prediction mechanism. This could not have been accomplished so quickly or easily given today's closed, proprietary network device processing. Another benefit is the fact that network packets now have the unprecedented ability to control their own processing. Great advantage was taken of this new capability in AVNMP. Virtual messages, varying widely in content and processing, can adjust their predicted values as they travel through the network. Finally, Active Networks add a level of robustness that cannot be found in today's networks. This robustness is due to the ability of AVNMP system components, which are active packets, to easily migrate from one node to another in the event of failure -- or the prediction of failure provided by AVNMP itself.

The desired characteristics of AVNMP are large a Lookahead time, high prediction accuracy, low overhead and robust operation. Each of these characteristics is inter-related and a suitable tradeoff needs to be determined during configuration of the system. The AVNMP experimental validation configuration for the initial test discussed in this paper is a feed forward network consisting of a host containing the Driving Process and four intermediate active network nodes containing Logical Processes as shown in Figure 86. AH-1 and AH-2 are host nodes and AN-1 through AN-5 are active network nodes. The edges between the nodes represent links between the labeled ports on each node. All nodes are Sun Sparcs running the Magician active network execution environment. The AVNMP system parameters were configured as shown in Table 8. In this experiment AVNMP is predicting the packet input and output rate for each link at each node, from an application residing on AH-1 that is transmitting an active audio packets.

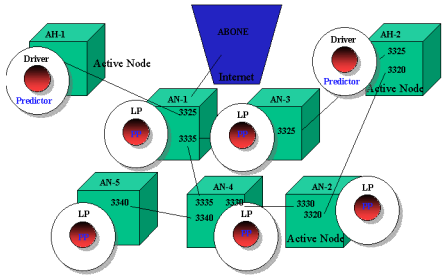

Figure 86. Experimental configuration.

Table 8 AVNMP Parameters

Sliding Window Lookahead Length	200 seconds
Virtual Message Generation Rate	0.5 virtual messages/millisecond
Virtual Message Step Size	20 seconds
Tolerance	500 Messages/second (reduced by half periodically)
Ratio of Virtual to Real Messages	1 virtual message/real message

2. Current project progress and experimental code is maintained in http://www.research.ge.com/~bushsf/ftn.

The State Queue plot, Figure 87, shows the predicted traffic load values cached in the State Queue as a function of LVT and Wallclock. As Wallclock approaches any given Local Virtual Time, the predicted load values converge towards the actual load. The general operation is illustrated in the next five graphs where all measurements, unless otherwise indicated, are from node AN-4. These curves validate intuitive trends in the operation of AVNMP. Figure 88 shows the reduction in tolerance versus time that is pre-programmed into each Logical Process. The Y-axis is the tolerance that is demanded between the predicted value and the actual value of an SNMP packet counter. This value is decreased purposely in this experiment in order to create a greater demand over time for accuracy and thus create a challenging validation of the AVNMP system under gradually increasing stress. In Figure 89 the proportion of out-of-tolerance messages is shown as a function of Wallclock. The Y-axis is the proportion of messages that arrived at a specific node out of tolerance, that is, the actual value exceeded the predicted value by an amount greater than the tolerance setting. As Wallclock progresses, the tolerance is purposely reduced causing a greater likelihood of messages exceeding the tolerance. This is done in order to validate the performance of the system as stress, in the form of greater demand for accuracy, is increased. Figure 90 shows the prediction error as a function of Wallclock. The Y-axis is the difference in the number of packets received versus the number of packets predicted to have been received. This graph verifies that the system is producing more accurate predictions as the demand for accuracy increases. However, the Y-axis of Figure 91 shows the Lookahead decreasing versus Wallclock. The expected Lookahead time is the difference between Wallclock and the Local Virtual Time at a particular node. The demand for greater accuracy reduces the distance into the future that the system can predict. Finally, in Figure 92, speedup, the ratio of virtual time to Wallclock of the real system, is shown as a function of Wallclock. The speedup is reduced as the demand for accuracy is increased. As previously mentioned, only for purposes of this experiment, the tolerance is being reduced as Wallclock progresses, causing the accuracy to increase while loosing performance in terms of speedup and Lookahead.

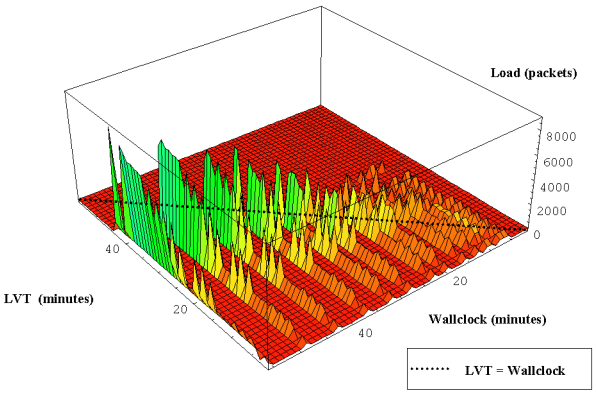

Figure 87. State queue.

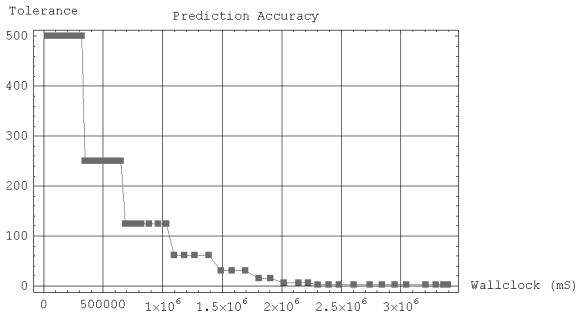

Figure 88. Tolerance setting decreases as wallclock increases thus demanding greater accuracy.

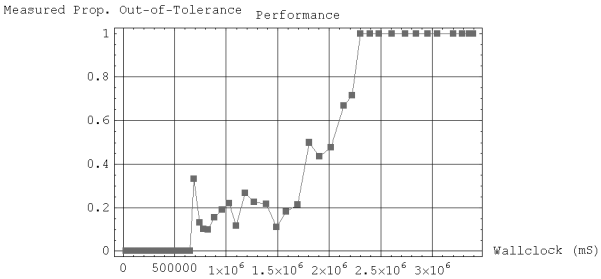

Figure 89. Demand for greater accuracy causes the proportion of out-of-tolerance messages to increase.

AVNMP overhead

AVNMP has the potential to generate two forms of overhead, processing overhead and bandwidth overhead. If the predicted results are within the user specified error tolerance and the user fully utilizes the predicted results, then overhead is at a minimum. The question of overhead versus benefit becomes one that depends upon the perceived utility of predictive capability and depends significantly

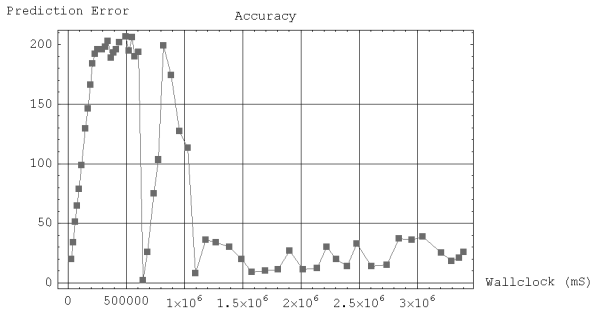

Figure 90. Predictions become more accurate….

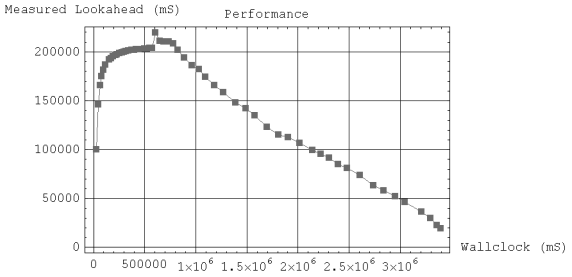

Figure 91. …at the expense of lookahead….

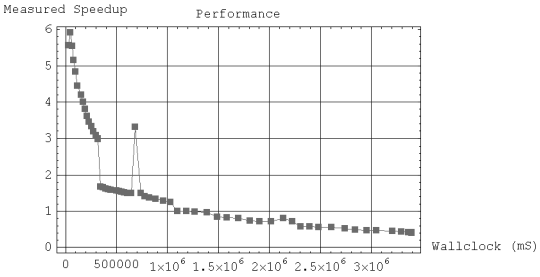

Figure 92. …and speedup.

upon the manner and application in which it is used. It is the author's belief that load and processing prediction are of particularly great importance in Active Networks where routing is based upon not only load, but the processing capability required by active applications. In this section, the load prediction application example is continued with overhead results displayed in terms of processing time and number of packets transmitted. The expected ANEP [3] packet size measured during the test was 1000 bytes.

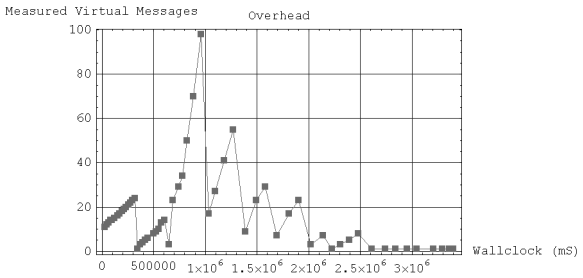

Figure 93. Number of virtual messages versus wallclock.

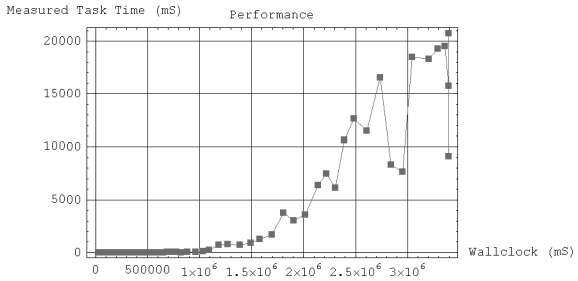

Figure 94. Expected task execution time as a function of wallclock.

Task execution time and message overhead

The task execution time is the Wallclock time the system spends executing a non-rollback message. It was expected that task execution time would be essentially constant; however, it increases in direct proportion to the number of rollbacks as shown in Figure 94. This is caused by the lack of fossil collection. The increase in the number of values in the State Queue is causing access of the State Queue and MIB to slow in proportion to the queue size. Figure 93 displays the number of virtual messages versus Wallclock and Figure 95 displays the total number of anti-messages. This is expected to increase over time. This value is reset every time the tolerance is tightened (every 5 minutes in this case).

AVNMP robustness

AVNMP consists of two main types of active packets: *AvnmpLP*, which is the Logical Process, and *AvnmpPacket*, which is the virtual message. If an *AvnmpLP* packet is dropped, the destination node will not have the capability to work forward in time or forward virtual messages. Thus, AVNMP features will not be available on the node and accuracy of other nodes may be reduced. If an *AvnmpPacket* is dropped or unexpectedly delayed, accuracy will be reduced

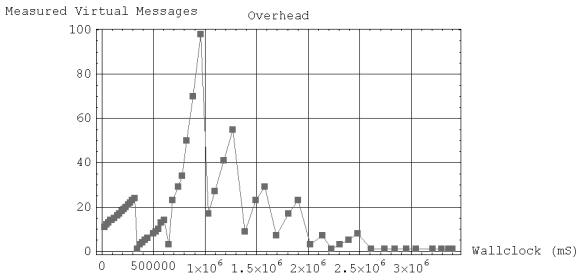

Figure 95. Number of anti-messages versus wallclock.

because the State Queues of downstream nodes will lack a predicted value. However, AVNMP will continue to operate. In the next section the role of complexity in understanding prediction is discussed.

4.2 TOWARDS COMPLEXITY

AVNMP can provide early warning of potential problems; however, the identification of a solution and marshaling of automated solution entities within an active network has not yet been fully addressed. This project has begun to lay the groundwork for such automated composition of management solutions within an active network [3]. This direction is being carried forward by exploration of a relatively unexplored area –understanding the benefits of active networking, Algorithmic Information Theory, and its close companion, Complexity Theory. To our knowledge, this work is the first to propose and begin investigation into the newly available processing power of Active Networks through the concept of Complexity and Algorithmic Information ("Streptichrons") as shown in Figure 96. Legacy networks, which are today's passive networks, have been designed to optimize transmission of passive data using bit compression based upon the underlying notion of Shannon Entropy. AVNMP has shown that active networks allow for the possibility of executable models and that the corresponding information packets might be best studied with Kolmogorov Complexity as the underlying theory. It is serendipitous that Complexity Theory has been receiving more attention lately and is making significant theoretical progress at the same time that research into active networking is taking place. Active networks provide a new paradigm and enhanced capabilities, which, when combined with ideas from Algorithmic Information Theory [10], might lead to superior, innovative solutions to problems of network management. One possible approach proposes to combine Kolmogorov Complexity with the science of Algorithmic Information Theory (sometimes called Complexity Theory) to build self-managed networks that draw on fundamental properties of information to identify, analyze, and correct faults, as well as security vulnerabilities, in a distributed information system [8],[9]. Specifically, we suspect that complexity measures can be used to detect and analyze problems in a network, and to facilitate techniques to remedy network faults. We also envision that Kolmogorov Complexity can be applied directly to improve the performance of AVNMP. In [5] and [6] the concept of monitoring the change in Kolmogorov Complexity of a system was first introduced for Information Assurance.

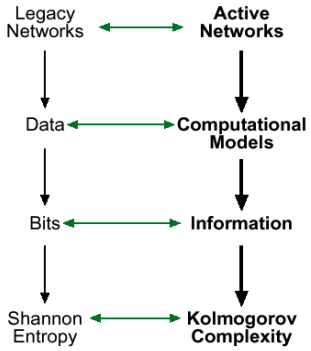

Figure 96. Active networks and legacy networks as viewed by AVNMP.

According to Complexity Theory, the complexity of an information unit is the size of the smallest program capable of producing the unit. Similarly, Algorithmic Information Theory defines the complexity of an information unit to be the unit's length (after the unit has been compressed to the maximum extent possible). These two views can be related through theory. In general, complexity is not computable; however, the bounds on complexity tighten continuously as fundamental research in Kolmogorov Complexity progresses. For example, the Minimum Data Length (MDL) [14] estimate for Kolmogorov Complexity considers that the best measure for complexity of an information unit minimizes the sum of the length of the description of a theory that produces the unit and the length of the unit encoded using the theory. In this section, we use MDL as one approach to estimate Kolmogorov Complexity, and we suggest its application as a means to improve the performance of AVNMP.

One potential drawback to AVNMP, gently pointed out earlier in this paper, is fact that AVNMP itself consumes resources in an effort to predict resource usage in a network. Resource consumption by AVNMP is tied directly to accuracy: higher accuracy costs more in terms of bandwidth utilization, associated with simulation rollbacks and the concomitant transmission of anti-messages. Despite this relationship, potential exists to nearly reach the theoretical minimum amount of bandwidth to achieve the maximal model accuracy. This possibility arises because AVNMP consists of many small, distributed models (each a description of a theory) that work together in an optimistic, distributed manner via message passing (data). Each AVNMP model can be transferred, using Active Networks, as a Streptichron [3], which is any message that contains an executable model in addition to data. Using Streptichrons, the optimal mix of data and model can be transmitted to closely approximate the minimum MDL. Achieving maximal model accuracy at minimal bandwidth provides the best AVNMP accuracy at the least cost in AVNMP resource consumption.

Other possibilities exist to exploit Kolmogorov Complexity to improve AVNMP performance. For example, one can apply the MDL technique to the rollback frequency of all the AVNMP enhanced nodes in a network. A low rollback complexity (which suggests a high compressibility in the observed data) would indicate patterns in the rollback behavior that could be corrected relatively easily by tuning AVNMP parameters. High complexity (low compressibility) would indicate the lack of any computable patterns, and would suggest that little performance improvement could be achieved by simply tuning parameters. Thus, we hypothesize that our tuning gradient should be guided toward regions of high complexity, which suggests that we can tune parameters to improve the rollback frequency. The next section focuses upon experimental results relating prediction to complexity gathered from the operation of the AVNMP system.

4.3 AVNMP AND KOLMOGOROV COMPLEXITY

In AVNMP, information that impacts the network is transmitted based upon prediction at a low level within the network. Thus, AVNMP allows experimentation in defining the boundaries within which active networking is beneficial. In Figure 97 an active and passive form of AVNMP is represented. The passive case is represented in the upper portion of the figure. In the passive case, actual data (D_x) is observed at the Driving Process. A hypothesis is formed about the data, and predicted data (D_y) is generated in the form of static virtual messages. The term static indicates that information content within the message contains no executable code. When error in the hypothesis exceeds a preset threshold, AVNMP causes rollbacks to occur in order to adjust for the inaccuracy. In the lower portion of Figure 97,

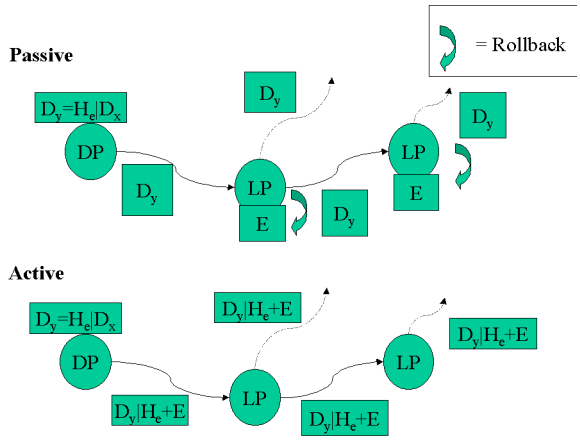

Figure 97. Active versus passive form of AVNMP.

the hypothesis is included within each packet and is used to encode within the code portion of the active packet.

What is the relationship between the estimated operating hypothesis (H_e) in the AVNMP packet encoding and as the predictor in the Driving Process? First, they are the same hypothesis. Second, it has been shown [10] that the shorter the packet, the better the predictor. Conversely, the worse the prediction, the longer the value is within the AVNMP packet encoding. Can Active Virtual Network Management Prediction benefit from the fact that the smallest algorithmic form is also the most likely predictor of a sequence? This can come about because Driving Processes and Streptichrons (active virtual messages anticipating events in the future) benefit by being both small and accurate as shown in Figure 98. The objective is to increase the rate of convergence of the predictions held within the State Queue to converge to the actual value that will occur in the future, and to converge to the value before it actually exists. Actual and predicted values within a particular instance of a State Queue were shown in

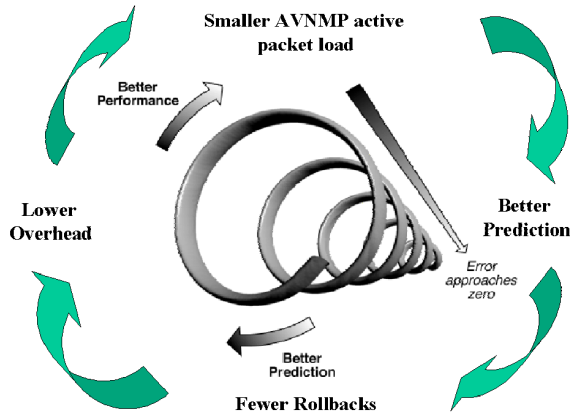

Figure 98. Better Prediction Implies Smaller Packets Implies Better AVNMP Performance Implies Better Prediction.

Figure 87. Let us examine A VNMP results in light of complexity in more detail in the next section.

Load prediction and complexity in active virtual network management prediction

Our Active Network Kolmogorov Complexity estimator is currently implemented as a quick and simple compression estimation method. It returns an estimate of the smallest compressed size of a string. It is based upon computing the entropy of the weight of ones in a string. Specifically it is defined in Equation 12

$$(\hat{K}(x) \approx l(x)H)\left(\frac{x\#1}{x\#1 + x\#0}\right) + \log_2(l(x)) \qquad (12)$$

where x#1 is the number of 1 bits and x#0 is the number of 0 bits in the string whose complexity is to be determined. Entropy is defined in Equation 13.

$$H(p) = -p\log_2 p - (1.0-p)\log_2 p - (1.0-p) \qquad (13)$$

See [7] for other measures of empirical entropy and their relationship to Kolmogorov complexity. The expected complexity is asymptotically related to entropy as shown in Equation 14.

$$H(X) \approx \sum_{l(x)=n} P(X=x)C(X) \qquad (14)$$

Load prediction data sampled from execution of AVNMP is analyzed relative to several hypotheses. The goal is to use a simple example to demonstrate the relationship among accuracy of hypotheses, complexity, and compression. The initial hypothesis (regardless of naiveté in choice of hypothesis) is that the data can be characterized by a simple linear

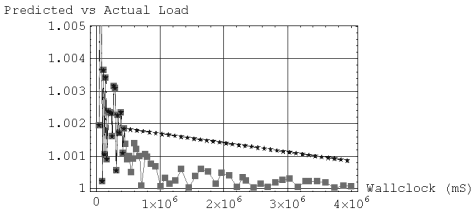

Figure 99. Load Prediction Hypothesis.

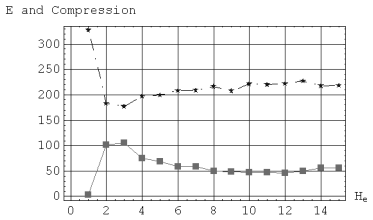

Figure 100. Simple AVNMP Hypotheses for Load Prediction.

extrapolation based upon the last sampled load values. This is shown in Figure 99 where the gray boxes are actual load samples and the black stars are predicted load samples. Note that the predicted load is based upon a short history shown in the graph as the initial match between predicted and actual load.

Various enhancements are added to the initial hypothesis. In this specific case, a running average was used to smooth the data before the extrapolation. The size of the running average defines a hypothesis. Each enhancement is considered a new hypothesis (H_e) in this experiment. In Figure 100, for each the sum of the error in predictions is graphed as the gray boxes in the lower portion of the graph. The compressed size of the corresponding error is plotted as the black stars in the upper portion of the figure. Clearly a better hypothesis concerning the origination of the data results in better prediction and greater compression, while poor hypotheses result in inaccurate prediction and reduced compression. This provides a concrete demonstration of the relation between complexity and prediction accuracy.

It is hypothesized that the greater the complexity, the greater the error in prediction, and thus the greater the likelihood of AVNMP rollback. In order to validate this hypothesis, load prediction error from AN-1 (see the experimental configuration shown in Figure 86) within the network is compared with the estimated complexity of the actual load. In Figure 101 the load prediction error is plotted with the estimated complexity versus Wallclock where val-

ues are taken over intervals of the same length as the Sliding Lookahead Window shown in Table 8. Larger error, and thus more likely rollback, occurs during periods of relative high complexity, while complexity is low during periods of low error.

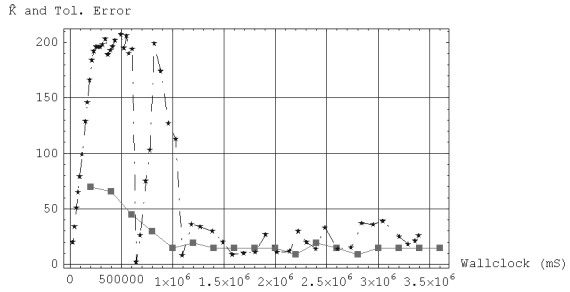

Figure 101. Estimated Complexity and Error within AVNMP.

Prediction convergence and complexity

Predictions within the State Queue form a sequence that AVNMP is trying to predict. This is represented in more detail in Figure 102. The goal of the rollback mechanism is to cause the predicted values to converge to the best-predicted estimate. In AVNMP, the Driving Process is the model, or MDL hypothesis. The virtual messages generated by the Driving Processes may be active, containing small hypotheses within themselves as previously discussed in the bottom of Figure 97.

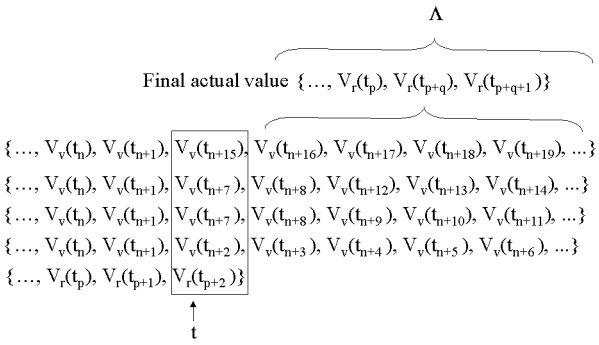

Figure 102. Converging Predictions.

The definition of Global Virtual Time, $GVT(t)$, can be applied to reason about the information contained in the State Queue. Consider task execution time (τ_{task}), which is the time taken by a logical process to generate a predicted value given an input message, the Wallclock time at which a particular state was cached (τ_{SQ}), and Wallclock time (t). Let P_{SQ} be the predicted time that event SQ will occur. Let $f(\tau)$ be the prediction hypothesis of a Driving Process such that $f(\tau)$ predicts a value for time τ where $\tau \geq t$. Consider a predicted value (V_v) that is cached at time in the State Queue resulting from a particular predicted event. As rollbacks occur, values for a particular predicted event may change, converging to the real value (V_r). For correct operation of Active Virtual Network Management Prediction, should approach as t approaches $GVT(t)$. Explicitly, this is $\forall \varepsilon > 0 \exists \delta : 0 < |f(t) - f(GVT(t))| < \varepsilon$ implies that $0 < |(GVT(t)) - t| < \delta$ where $f(t) = V_r$ and $f(GVT(t)) = V_v$, Because Active Virtual Network Management Prediction always uses the correct value when the predicted time (p) equals the Wallclock (t) and it is assumed that the predictions become more accurate as the predicted time of the event approaches the current time, the reasonable assumption is made that $\lim_{t \to p} f(t) = V_r$. In order for the Active Virtual Network Management Prediction system to always look ahead, $\forall t$: $GVT(t) \geq t$. This means that $\forall n \in \{LP\} t \in LVT_{lp_n}(t) \geq t$ and $\frac{\min}{m} \in \{M\} \geq t$ where m is the receive time of a message, M is the set of messages in the entire system and LVT_{lp_n} is the LVT of the n^{th} Logical Process. In other words, the Local Virtual Time of each process must be greater than or equal to Wallclock and the smallest message not yet processed must also be greater than or equal to Wallclock. The smallest message could cause a rollback to Wallclock. This implies that $\forall n, t: LVT_{dp_n}(t) \geq t$. In other words, this implies that the Local Virtual Time of each driving process must be greater than or equal to Wallclock. An out-of-order rollback occurs when $m < LVT(t)$. The largest saved state time such that $P_{SQ} < m$ is used to restore the state of the Logical Process, where P_{SQ} is the time the state was predicted to occur. Then the expected task execution time (τ_{task}) can take no longer than $P_{task} - t$ to complete in order for $GVT(t)$ to remain ahead of Wallclock. Thus, a constraint between expected task execution time (τ_{task}), the predicted time associated with a state value (P_{SQ}), and Wallclock (t) has been defined. As H_e improves there will be a reduction in the number of rollbacks, a smaller value in the packet encoding, and shorter Streptichrons.

Self-regulation via complexity

As predictions become more inaccurate in AVNMP, virtual messages should slow down, rather than burden the system with potential rollbacks. Poorly predicted messages will naturally be larger in their

minimum size, which slows down their rate of propagation in proportion to their inaccuracy.

Another issue concerns a mechanism for feedback to the Driving Process in order to improve. Such a feedback mechanism can be based upon input from the complexity estimate, or minimum encoded packet size, of virtual messages. The hypothesis is adjusted in a manner that drives the system towards minimizing encoded virtual message size.

5. Fault Identification and Extraction

Active Virtual Network Management Prediction (AVNMP) [16], provides predicted state within each node of a network based upon a correct estimated operating hypothesis, H_e. The AVNMP architecture provides a network prediction service that utilizes the capability of Active Networks to easily inject fine-grained models into a communication network to enhance network performance. AVNMP, injected into the network as an active overlay network, is a simulation of the actual network, but running temporally ahead of the actual network. AVNMP is capable of modeling load and propagating state information in a manner that meets the demand for prediction accuracy at a particular active node at the expense of overhead due to rollback in order to correct for prediction inaccuracy. Thus, prediction accuracy is met at the cost of AVNMP performance, that is, the ability of AVNMP to predict farther into the future. An AVNMP application to predict CPU utilization for Active Networks has been demonstrated in collaboration with National Institute of Standards and Technology in [17], [35] and [36]. The inherently distributed nature of communication networks and the computational power unleashed by the Active Networking paradigm have been used to mutual benefit in the development of the Active Virtual Network Management Prediction mechanism. Active Networks benefit from AVNMP by continuously receiving information about potential problems before they occur. In this paper, the groundwork is laid for using a Kolmogorov Complexity estimate to drive the optimal generation and composition of solutions. System faults are represented in algorithmic form. Reversible code is then developed to remove the effect of faults in a system. The application in this paper focuses on an active network in which information, algorithmic and static, can be transmitted in a fine-grained manner.

Kolmogorov Complexity ($K(x)$) (see [30] for an introduction to Kolmogorov Complexity and [18], [21] [28] and [29] for applications) is the optimal compression bound of string x. This incomputable, yet fundamental property of information has vast implications in a wide range of applications including network and system optimization, security, and Bioinformatics. An active network [16] provides a suitable environment in which to study the effects of tradeoffs in algorithmic and static information representation because an active packet consists of both code and static data. An active network enables packets to perform computation at the intermediate nodes in addition to its communication capabilities. This enables application developers to design novel network services and protocols that can trade-off communication and computation as the packet traverses the network. Therefore, to ensure best performance, active network developers have to effectively answer the question, “What is the optimal proportion of the code size in a packet with respect to its data payload?” A method for obtaining the answer to this question comes from direct application of a technique called Minimum Description Length Minimum Description Length (MDL) [37],[30] to an active packet. Let be a string of bits representing x. Let be a hypothesis, or algorithm, that attempts to explain how is formed. MDL states that the sum of the length of the shortest encoding of a hypothesis about the model generating the string and the length of the shortest encoding of the string encoded by the hypothesis will estimate the Kolmogorov Complexity of string x. A method for determining separates randomness from non-randomness in x by incorporating non-randomness, which is computable, as the shortest encoded program that represents the original string. The random part of the string represents the error, that is, the difference between the original string and the output of the encoded program. Thus, the goal is to minimize where is the length of string x, is the estimated hypothesis used to encode the string () and is the error in the hypothesis. The more accurately the hypothesis describes string x, the shorter the encoding of the string. An active packet is measured as shown in Figure 103, where each packet conveys the same information; however, the length varies with the choice of the proportion of code and data. This in turn is governed by the hypothesis chosen to represent the data. The better the hypothesis, the lesser the “error” made in representing the data. In the figure, H1 is the worst hypothesis as the “error” and hence the packet size is the largest. On the other hand, H4 presents the best hypothesis for the data and hence the packet size is the smallest. The goal is to learn how to optimize the combination of communication and computation enabled by an active network. Clearly, if is estimated to be high for the

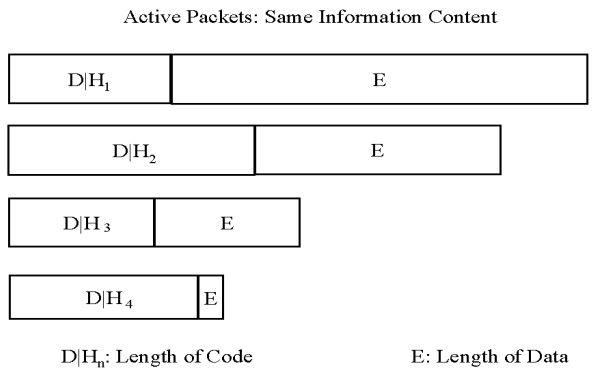

Figure 103. Algorithmic content.

transfer of a piece of infor mation, then the benefit of having code within an active packet is minimal. On the other hand, if the complexity estimate is low, then there is great potential benefit in including it in algorithmic form within the active packet. When this algorithmic information changes often and impacts low-level network devices, then active networking provides the best framework for implementing solutions. In the next section, the relationship among AVNMP, Kolmogorov Complexity, and fault behavior are examined in more detail.

5.1 AVNMP AND FAULT PREDICTION

The Active Virtual Network Management Prediction mechanism (AVNMP) [16] requires the injection of models, or hypotheses, that describe the operation of the system assuming no fault exists. An example derivation of a non-fault-operating hypothesis and its relationship to Kolmogorov Complexity is discussed in [155]. When Wallclock time reaches a predicted state time, verification is made to determine whether the predicted value deviates beyond a preset tolerance from the actual value. If the prediction is accuracy fails to fall within the tolerance, an out-of-tolerance rollback occurs. Out-of-tolerance rollbacks in AVNMP are due to inaccurate prediction and thus are related to error, or inability of the hypothesis in MDL to fully capture all patterns in a string. The rollback mechanism, which is also correlated to the length of *E*, Figure 103, in the encoded packet as discussed in [155], accounts for randomness. That is, randomness is defined as information incapable of being compressed algorithmically and cannot be defined algorithmically, and thus cannot be predicted. This results in a higher Kolmogorov Complexity as experimentally validated in [155].

However, a fault occurring in the system will appear as a deviation from the non-fault-operating hypothesis. The fault will induce the appearance of greater randomness, or higher Kolmogorov Complexity because actual events will not fit the initial estimated hypothesis (H_e). This will cause an increase in rollback frequency and a longer value in the encoded packets. Most Bayesian Belief Networks take the opposite approach by developing fault hypotheses (H_f) rather than a correct operating hypothesis. Clearly an approach based upon handcrafting fault hypotheses assumes one can predetermine and characterize all possible faults, a large and difficult task. The goal of this project is *self-healing*, in which the system automatically aligns itself with the correct operating hypothesis in the presence of unanticipated faults.

A potential complication that arises when AVNMP is viewed in this manner is that is, by definition, only an estimate of the correct operation of the actual system. Rollbacks, randomness and occur as a result of the deviation of H_e from H, where is the true and complete hypothesis describing the system. The question arises as to how to distinguish between randomness due to a faulty operating hypothesis and an actual fault. Figure 104 illustrates the dichotomy. In

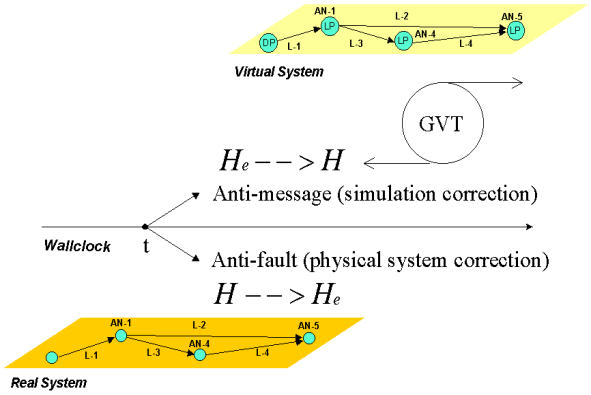

Figure 104. Self-correcting simulation versus fault correction within the actual system.

the upper portion of the figure, the virtual system runs ahead of Wallclock time such that Global Virtual Time (GVT), that is the estimate of time to which the entire AVNMP system has advanced, proceeds at a faster rate than Wallclock. The physical system, in the lower portion of the figure proceeds at the rate of Wallclock. In order to correct the virtual system, anti-messages, in the form of reversible code [20], can be transmitted. This reduces the for-

ward execution rate of the vir tual system in an attempt to bring the virtual system, based upon an estimated hypothesis (H_e), closer to the actual system operation described by hypothesis H. Alternatively, in the real system, anti-faults in the form of reversible code can be generated to move the actual system () towards the estimated hypothesis (H_e). The next section considers anti-faults in more detail.

5.2 ALGORITHMIC FAULT DETECTION AND GENERATION

There are at least three reasons why an algorithmic description of a fault is desirable. First, constructing the smallest algorithmic representation of a fault indicates its complexity, which is valuable information. Complexity is important information because it is an indicator of both the type of fault and level of difficulty in correcting the fault and the severity of the fault; fault severity is important in triage operations to optimize system health. Second, a more compact algorithmic representation of a fault will travel faster and more rapidly through the network; it is an efficient format for alerting system management and in triggering automated solutions. Third, it is relatively easy to reverse the code of an algorithm, possibly generating an anti-fault, or solution to a problem in certain cases. Reversible code has been presented in previous work as a mechanism for generating anti-messages in Time Warp simulation. In this section the behavior of complexity with regard to code and anti-code is discussed as well as results leading towards the use of reversible code for self-composing solutions.

The proposed hypothesis is that the Kolmogorov Complexity of a combined fault and solution description is minimized when the optimal solution to mitigate the fault is composed. A nearly trivial example can be seen with reverse code. Assume that fault data, F exists. Assume that the fault does not erase any data but merely transforms it. Define the algorithmic description of the fault data $P_F()$. The reverse code for $P_F()$ will be labeled $RP_F()$. Assume $P_F()$ and $RP_F()$ are minimal length programs. Then, $RP_F(P_F()) = \phi$ where ϕ is the empty set. RF is the data generated by $RP_F()$. Since the fault does not erase any data, the process is reversible [30] and therefore, or $K(RF) - K(F) = 0$. The equivalence in complexity RF and F follows from the fact that because there is no loss or gain of complexity when the system is restored to its prior state using the anti-fault process RP_F, there is no work performed. The algorithmically reversed fault will be referred to as an anti-fault in this paper.

Consider reversing AVNMP processes in more detail. Details of AVNMP operation are described in [16] however, a brief description is provided here using a Case Diagram. The Case Diagram shown in Figure 105 describes the AVNMP Management Information Base (MIB). Arrows indicate information flow; labeled short lines indicate counters, and labeled arrows represent information flow that is counted. Active packets arrive through the active channel to the AVNMP Logical Process. The total number of packets entering the Receive Queue is maintained in *logicalProcessQRSize*. In addition, the Local Virtual Time is sampled via *logicalProcessLVT*. Next the State Queue, which holds past, present, and predicted state values, is updated. Any rollbacks that may occur are counted. Next the Send Queue transmits any output messages generated by the AVNMP Logical Process. The logicalProcessAntiMessages counter counts anti-messages separately.

Now consider the effect of reversible code using the SNMP Case diagram previously discussed in Figure 105. First, it is important to distinguish between the Physical and Logical Process. The Physical Process is the model of the actual system injected into AVNMP. This is in contrast to the Logical Process, which is the entire AVNMP supporting implementation that includes the Physical Process as well as possible state saving, rollback, and anti-message capabilities. Note that it is the Physical Process, that is, the object being modeled that must be reversed, not the AVNMP Logical Process described in Figure 105. Case Diagrams shown in Figure 106 and Figure 107 represent the Physical Process. A process operating in reverse would be required to effectively

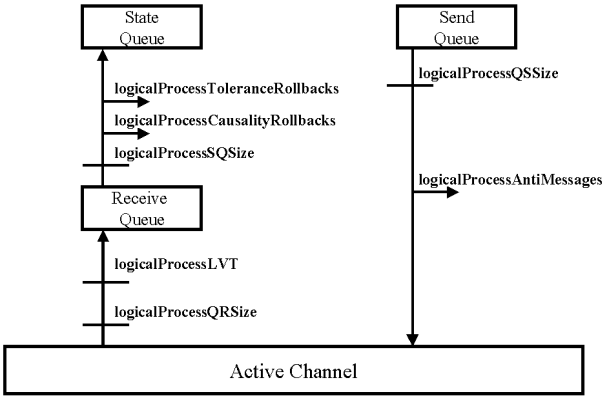

Figure 105. AVNMP SNMP Case Diagram.

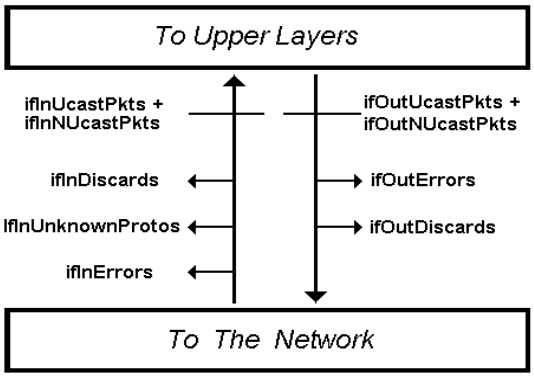

Figure 106. Case diagram for IP.

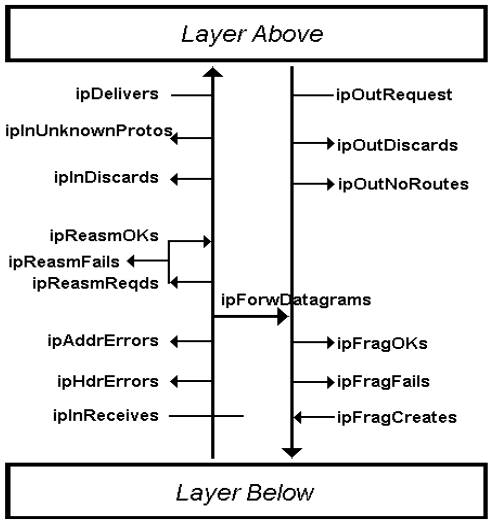

Figure 107. Statistics maintained for each interface.

reverse all ar row directions resulting in an effective decrement of counters. This is the purpose of rollback and anti-messages. By chasing the original message, the anti-message may cause additional rollbacks to occur which, in a state saving system, causes previous state values to be restored to a known valid state. When implemented using reversible code, the anti-message actively undoes the effect of the original message(s). The original messages to be rolled back are input to a reverse code version of the physical process in reverse order of their Receive Times. Instead of reversing a process that persistently resides on a node, one could imagine reversible active packets. The next section discusses a specific application of anti-faults, pointing out potential disadvantages with such a technique.

5.3 DISTRIBUTED DENIAL OF SERVICE EXAMPLE

In the previous section, it was suggested that general-purpose fault correction might be automated by reversing the algorithmic fault description. This section considers the effect of such a mechanism relative to a particular fault, namely a Distributed Denial of Service (DDoS) attack. We will see that simple fault reversal may not always be the best solution, particularly when irrevocable events have occurred, such as the theft of resources. Consider a Distributed Denial of Service (DDoS) attack as a fault. Kolmogorov Complexity has been used to detect likely DDoS attacks in [154]. Note the assumption that this is an Active Network, thus, the DDoS attack can consume both bandwidth and processing. Referring back to Figure 104 one can see that AVNMP continuously updates predictions on anticipated load throughout the system based upon legitimate network use. In the situation illustrated in Figure 106, a simplified snapshot of packet forwarding is illustrated in which packets entering network interfaces *x, y* and *z* are forwarded outward through interfaces *a, b, c.* The Case Diagrams in Figure 106 and Figure 107 represent a more detailed model of information flow. Remember that these flows describe the Physical Process as mentioned in the previous section. As protocol data units move through the network interfaces, the Simple Network Management Protocol (SNMP) counters shown in Figure 108 will maintain current state in separate

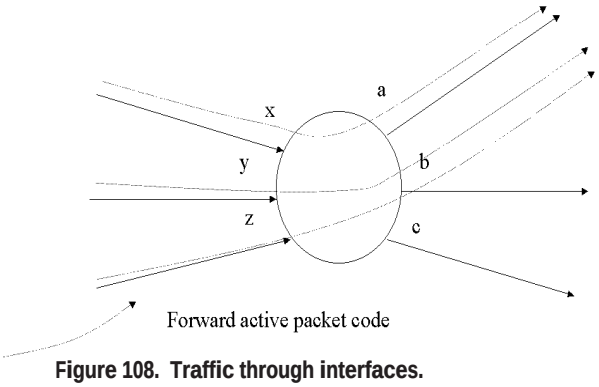

Figure 108. Traffic through interfaces.

MIB table rows for each interface.

A hypothetical set of traffic load graphs is shown in Figure 109. A simple algorithmic form of the load might be as shown in the figure at time *t*, namely $a = x + 2y + 10z$ where *x*, y, and *z* are output inter-

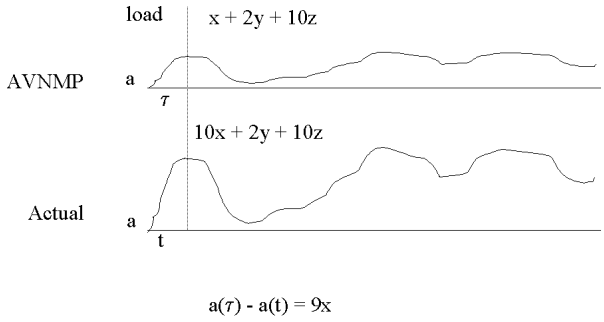

Figure 109. Predicted versus actual load.

faces and the numbers indicate load, in number of packets, on those interfaces. The fault data (F) is the difference between the load in the AVNMP and Actual graphs. The algorithmic description of the fault ($P_F()$) is the code that forwards nine packets from interface x to interface a. The reverse code ($RP_F()$) would transmit 9 packets from a back to x, essentially reflecting the attack back towards the source as illustrated in Figure 110. The authors rec-

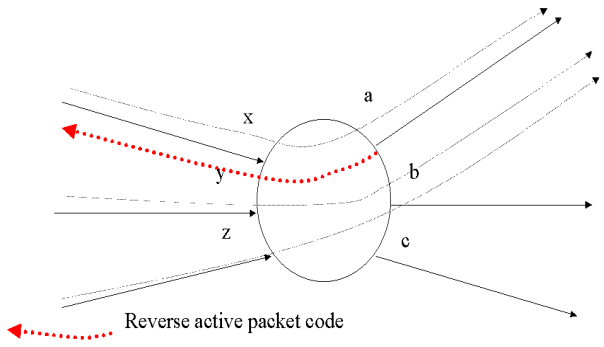

Figure 110. Reflecting the attack via code reversal.

ognize that simply reflecting packets would increase network load, effectively increasing the impact of an attack. This is a disadvantage of blindly using anti-faults. In this case, it is disadvantageous to blindly use anti-faults because the fault process performed the irreversible actions of using up bandwidth as well as processing units. The best option in this case might have been to simply attempt to quench attack packets while moving closer towards the source of the attack. This would bring the actual system closer to the expected operation provided by the AVNMP model, namely $H \rightarrow H_e$.

The goal of the anti-fault was to place the system back to a healthy state that existed before the fault occurred. The naïve anti-fault described above, while relatively easy to implement, attempted to do this by reversing events, some of which were irreversible. For example, once a resource, such as bandwidth or CPU has been stolen at a particular instance of time, it cannot be returned. Additionally, the attempt to transition the System State backward in time to a healthy condition temporarily increased the impact of the fault. Research required to achieve the effective reversal of faults in a more controlled manner using complexity is outlined next. A Swarm simulation of system complexity is used to study the relation to Kolmogorov Complexity.

5.4 TOWARDS COMPLEXITY-BASED SOLUTION COMPOSITION

This section discusses a general approach for self-composing solutions using lessons learned from the previous section. The approach can be described as the automated generation of a solution hypothesis $H_s = R(H_e - H_f)$, i.e., the reverse of the algorithmic difference between the faulty and correct algorithmic representation of behavior by controlled means. As H deviates from, heat [28],[29] or complexity as presented here, is generated. In [28] and [29] the relationship between fault and energy is explored and simulated (see [18] and [21] for recent work on complexity and energy and Information Assurance). It is useful to briefly describe [28] in order to provide a tangible background and explanation for the algorithmic fault detection approach. The motivation for that experiment came from the relationship between Kolmogorov Complexity and entropy. The definition and application of Kolmogorov Complexity to vulnerability analysis (discussed in [21]) identified how Kolmogorov Complexity can be used to determine vulnerabilities in a system as areas of low complexity. An underlying hypothesis of our work is that computation and communication are fundamentally related and conversely bandwidth and processing denial of service are fundamentally interrelated. Low complexity data or code consuming large amounts of bandwidth or processing indicates the likelihood of an attack. A model of complexity evolution within a closed system is described in reference [18]. That reference developed an abstract model with which to study complexity, specifically Kolmogorov Complexity, of information within an information system. That

```
In[1]:= Needs["Complexity`KEstimates`"]

In[2]:= ? *Complexity

Complexity`KEstimates`
AutoComplexity  StringAutomatonComplexity  StringComplexity

    StringComplexity[s, opts] returns an estimate of the complexity of a bit string.

In[3]:= ? *Compression

    Compression[i] returns the compressed size of a binary list of bits.
```

Figure 111. Selected mathematica complexity functions.

model explores *K(x)*, a measurement of length in bytes, and *K(x)/s*, a measure of the maximum increase in complexity of the system due to code entering a system such as code carried by active packets. The rate of complexity increase in terms of algorithmic active packet complexity in units of within the closed system was measured. Significant changes in system complexity indicate the presence of faults. Reference [154] reported the results of Kolmogorov Complexity probes that detect Distributed Denial of Service attacks.

Complexity estimation mechanisms for these experiments have been developed in Mathematica using a package developed specifically for the study of complexity, particularly within active networks. This package contains several functions for the estimation of complexity, shown in Figure 110, including a finite automata minimization technique and an entropy-based compression technique. The package also contains a framework for simulating transmission of data in user controlled combinations of algorithmic and passive forms within active packets. After testing in Mathematica, the implementation has been integrated into an active network [16] as Java code that can be easily inserted into Magician [31] active packets. The complexity probe returns an estimate of the smallest compressed size of a string. It is based upon computing the entropy of the weight of ones in a string. Specifically it is defined in Equation 15A where *x*#1 is the number of 1 bits and *x*#0 is the number of 0 bits in the string whose complexity is to be determined. Entropy is defined in Equation 15B. The expected complexity is asymptotically related to entropy as shown in Equation 15C. See [153] for more advanced measures of empirical entropy and their relationship to Kolmogorov Complexity. The expected complexity is asymptotically related to entropy as shown in Equation 15C.

$$(\hat{K}(x) \approx l(x)H)\left(\frac{x\#1}{x\#1 + x\#0}\right) + \log_2(l(x))$$

A

$$H(p) = -p\log_2 p - (1.0-p)\log_2 p - (1.0-p)$$

B

$$H(X) \approx \sum_{l(x)=n} P(X=x)C(X)$$

C (15)

In references [18] and [28], a Swarm simulation containing agents representing data points from a system, specifically, Simple Network Management Protocol (SNMP) Management Information Base (MIB) object values from a communication network were programmed to initially move in a randomized manner. Thus, the agents began with high location entropy and no detectable pattern formation resulted initially. This mechanism was used to maximize the ignorance of the prior probability, which is also the purpose of the universal probability, *M()* [27]. The location distribution of Swarm agents, *x*, represented the health of the system. The predictive capability can be viewed as the ability, given *x*, to predict the type and severity of a fault condition pattern, *y*. The goal is to predict *M(y|x)*. From Bayes Theorem [30] the problem can be stated as shown

in. Figure 112 shows the relationship among

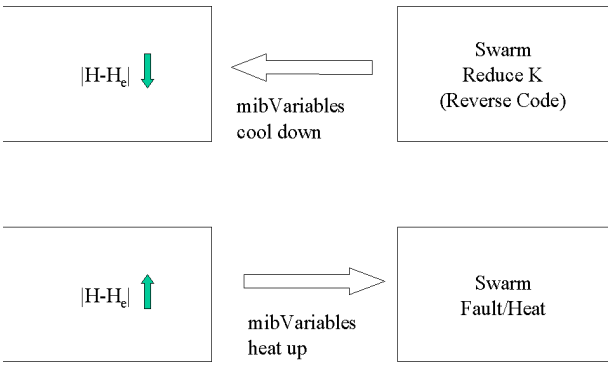

Figure 112. Hypotheses, complexity, and entropy in anti-fault generation.

AVNMP hypothesis deviation, entropy, and complexity in the feedback mechanism designed to maintain system health.

$$M(y|x) = \frac{M(xy)}{M(x)}$$

Equation (16)

Faults, representing MIB object values that operated outside of a preset threshold generated heat proportional to the amount by which they exceed the threshold, $|H_e - H|$. The agents were attracted towards the location of heat. An accurate heat propagation model was used in the simulation to model heat dissipation within a finite two-dimensional grid upon which the agents resided. With the generation of heat, the agents moved in a consistent direction towards the heat, and then clustered together in a circular pattern around the heat resulting in a loss of entropy. In a sense, the introduction of entropy via heat energy caused a reduction in entropy of agent location. Equation (17) and Equation (18) indicate the most accurate prediction for fault pattern location distribution y is one that minimizes the difference in length between the program that generates x and the program that generates xy. Clearly, in the perfect operation scenario, movement was programmed by default to be as randomly generated as possible. The program size required to define the location of each agent was on the order of the size of the entire grid. As heat was increased, cluster patterns increased in number and size, causing the location distribution to be describable by smaller formulae, thus lower complexity. The cluster patterns were hypothesized to represented the type of fault while the complexity, or size of the algorithmic description of the cluster patterns, estimated the severity of the fault.

$$\log M(y|x) = \log M(xy) - \log M(x)$$

Equation (17)

$$\lim_{x \to \infty} -\log \mu(y|x) = Km(xy) - Km(x) - ()(1)$$

Equation (18)

The Swarm complexity model is only as good as its ability to reflect complexity changes in an actual system. An actual system contains many subtle correlations that may be impossible to fully specify in detail. However, let us begin with a controlled experiment involving a DDoS attack. Swarm agent location over time represents complexity in this experiment. Swarm heat is a control mechanism that effectively introduces correlation and reduces complexity. The goal is to model information flows in which Swarm agents represent blocks of time averaged active packets in the DDoS differential complexity window described in [154]. Heat represents potential correlation that indicates potential DDoS attack. We begin by calibrating Swarm heat parameters to match known Magician DDoS complexity estimations. The attacker varies the correlation within the Magician attack stream. The Swarm model is correspondingly varied and the results compared and contrasted with the actual Magician system.

5.5 SUMMARY

A Kolmogorov Complexity estimate is used to drive the optimal generation and composition of solutions. System faults are represented in an algorithmic form. Reversible code is then developed to remove the effect of faults in a system. The application in this paper focuses on an active network in which information, algorithmic and static, can be transmitted in a fine-grained manner.

6. Information Assurance

6.1 ANALYSIS OF THE EVOLUTION OF COMPLEXITY

A defender would like to know not only the average complexity of the system, but also the change in complexity over time. The goal of this section is to attempt to address the macroscopic behavior, or evolution, of complexity in order to understand how complexity relates to vulnerability and the ever-increasing cycle of attacker / defender complexity as each improves their capabilities. Even if complexity can be measured at various locations within an information system as discussed later in this report, the system may evolve. It is critical to know if bounds on complexity evolution exist so that complexity "probe" locations can be optimized. Understanding the evolution of complexity can also lead to optimal sampling of the probes so that they are not over-sampled, causing wasted network resources.

Using Definition 6.4, the evolution in complexity can be crudely characterized by placing initial estimated values into Definition 6.4 and recursively computing the complexity of the output $y = p(x)$. The output, y, is a data bit-stream and can sometimes be an executable program, $p()$. The resulting evolution is a series of the form $\{y\}$, where. The characterization of this series depends upon the initial values in Definition 6.4, namely $L(P)$, $K(x)$, c, and the magnitude of the inequality from which $K(y)$ is derived.

In order to provide convenient measurements for comparison with the macroscopic results discussed later in this report, the metrics are defined in a manner independent of factors influenced by the operation of the system itself. These include such parameters as wallclock time, number of program or data bit-strings, or initial location, rate, and direction of movement of entities within the system. The metrics analytically derived from Definition 6.4 include, $E[K(y)]$, $E[L(y)]$ where K is the estimated Kolmogorov complexity and $E[\,]$ is the expected value. The number of cycles, or evolutions, of Definition 6.4 is used to control the termination of the analysis. In the emulation described later in this report, a program can terminate early or continue executing forever; both are the results of input not being what is subjectively called a valid program. This analysis assumes that neither event happens. Both events result in fewer bit-strings in the actual system than in the analysis. The manner in which these events are handled in the actual emulation is discussed in more detail later in the report.

The analytical results obtained are shown in Figure 113 using Definition 6.4 where the constant

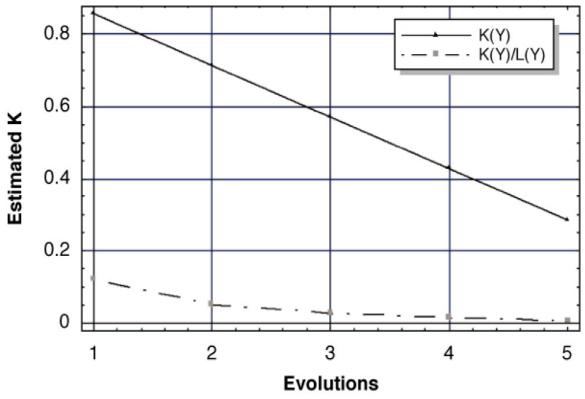

Figure 113. Definition 6.4 with Estimated Complexities for the Data and Program.

and inequality are determined from a compression-based estimate of a single specific program, input, and output complexity measurement. The more inefficient the program, where program efficiency is $\frac{K(p)}{L(p)}$, the more rapidly the complexity can evolve per program execution. The specific program whose complexity estimates are used in Figure 113 are from the program used in the emulation described later in this report.

Given the means to monitor estimates of complexity within an actual, highly dynamic, and evolving information system, trends that support the theoretical bounds on complexity should be observable. The next section attempts to construct such a system by obtaining measurements of bit-string lengths, complexity, and Turing Machine program transitions executed in a highly dynamic environment in which programs, data, and machines come together sharing information in a large-scale system.

Emulation of complexity evolution

The effort to implement a model of the evolution of complexity is called emulation rather than simulation because the computation and complexity under analysis are part of the model itself. The primary goal of this emulation is to examine trends in the

evolution of complexity in a highly complex interconnection of information systems. In particular, an initial set of carefully controlled environmental parameters should include initial program and data complexities and lengths as well as the rate of exchange and transport of information. Given an initial set of programs and data bit-strings let loose to execute in a closed environment, how does the complexity evolve? At what rate does complexity increase and why? What affects its rate of increase and why? What would cause it to decrease, at what rate, and why? What is the relationship between the microscopic level, that is, a single program execution, and the macroscopic level? What is the relationship between the complexity of a program and the complexity of the output it generates? Does the microscopic change in complexity hold at the macroscopic level and can more detail be deduced about the bound on complexity that it describes? What is the relationship of program execution, that is, Turing Machine state transitions, to output complexity? How do controls on information exchange affect the evolution of complexity? How does a set of programs, some that increase complexity and some that decrease complexity, affect the evolution of the total system complexity? This work attempts to lay the groundwork for answering these questions by comparing and contrasting the microscopic and macroscopic levels of complexity.

The system used for experimental study of complexity should mirror a large-scale, highly dynamic, yet easily controlled environment that is representative of actual information **exchange** and **evolution** in a variety of real world information systems. The approach taken in this effort to understand the evolution of complexity is illustrated in Figure 114. A Swarm (http://www.swarm.org) model has been developed that provides an open framework for experimentation on Complexity Theory and its relationship to information assurance. The Swarm emulation has been programmed with three types of entities: Turing Machines, programs, and data bit-strings. Multiple entities of each type exist and are represented in Figure 114. The **program** is represented by a set of states connected by transitions, the bit-string **data** by binary digits within a rectangular array, and the computer chip represents the Turing Machine. Each entity is placed in a random location within a field of attractive force, causing data bit-strings to be attracted towards programs and programs towards **Turing Machines**. The system can be

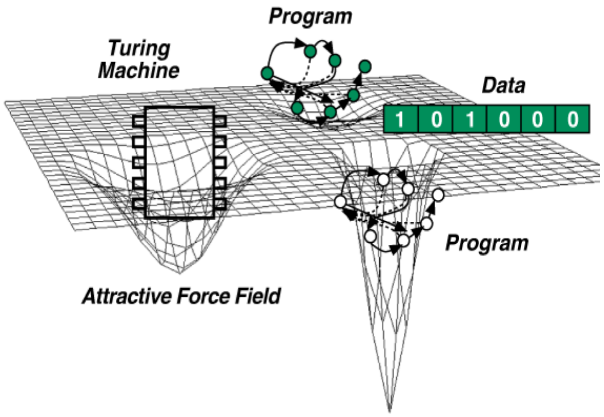

Figure 114. Emulation Components and High Level Dynamics.

viewed as a two-dimensional grid shown in Figure 115 with a possibility of four types of objects

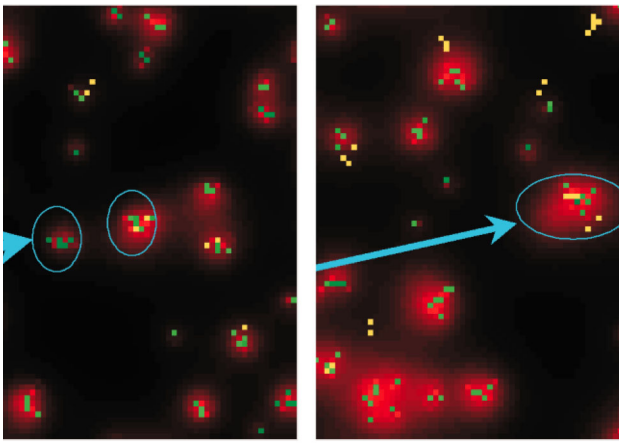

Figure 115. CyberSwarm Simulation at 100 and 300 Time Units.

residing on any grid location, as shown in Table 9. Note that all interactions and control decisions are made by the local entities; there is no global control.

Table 9 Entities and Their Representation in the Emulation

Entity	Representation
Turing Machine	Yellow—only when active
Tape	Green—lighter shading when more complex
Turing Machine Program	Green—lighter shading when more complex
Heat	Red—Darker when hotter

The motivating factor for each of the entities is heat. Each entity generates a small amount of heat that is determined probabilistically within a range specified as a startup parameter. Heat can represent the initial flow of information that causes one entity to become interested in working with another and form groups that are not too small or too large. Using energy to represent information is discussed further in [24]. The mechanism that implements this clustering behavior is the fact that entities are endowed with a desire to maintain an ideal temperature and will cluster together via a randomized movement pattern in a direction seeking to maintain their ideal temperature. Heat diffuses through the system in a realistic manner. The brightness indicates the estimated complexity of an entity – dark green indicates a low complexity, brighter green indicates a higher complexity. In order to provide a preview of how this emulation is designed, Figure 115 shows the emulation at times 100 and 300 units respectively. These figures should be viewed in color in order to see the full interpretation. If a Turing Machine, program, and data input tape meet, the program is executed and the program output tape generated by program execution is added to the system – available for use as data input in further computations. Figure 115 highlights two clusters: the circled cluster on the left is dark because the bit-strings have low and no Turing Machine to enable computation while the right cluster is brighter because it contains more computational activity and growing per bit-string. The clusters are attracted to one another and eventually form a single larger, brighter cluster.

There are two levels of abstraction occurring simultaneously in this system. The first level is individual goal-directed behavior and information exchange; the second is computation. In the first level of the abstraction, heat is a representation of motivation for movement towards common areas. The entities are initially located randomly throughout the two-dimensional space and gradually cluster together seeking to reach an ideal temperate. There is complexity in the location and movement of the entities. Movement towards common areas allows information exchange to occur. At this level of abstraction, concepts such as access control in the exchange of information can be explored; however, that is outside the scope of this report. The second level of abstraction is focused on computation. That is the primary focus of this report. The first level of abstraction provides the highly dynamic system in which the second level, computation, takes place.

The Turing machine

The Turing Machine entity enables a program consisting of sets of states and transitions to operate. The Turing Machine is one of the most fundamental general computing abstractions and is well-known in computer science, having a rich theory of its own that this report intends to utilize to its advantage. The Turing Machine consists of a seven-tuple (Q, T, I, δ, b, q_0, q_f). Q is a set of states, T is a set of tape symbols, I is a set of input symbols, b is a blank, q_0 is the initial state, q_f is the final state. δ is the next move function. δ maps a subset of $Q \times T^k$ to $Q \times (T \times \{L, R, S\})^k$. L, R, and S indicate movement of the tape to the left, right, or stationary respectively. There can be multiple tapes. Given a current state and tape symbol, δ specifies the next state, the new symbol to be written on the tape, and the direction to move the tape. The sets of symbols that lead to an accepting state (q_f) is the input language ($\sum$). The Turing Machine object, when loaded with a valid program, can execute the program when provided with an input tape. A program resides on a tape; the only difference between a program and a tape is that a program is a set of instructions that the Turing Machine can interpret as a program. If the program fails, that is, it is not a syntactically valid program, the Turing Machine will eject it. If the time taken by program execution on a Turing Machine exceeds a specified time limit, the program is forcibly terminated. The Turing Machine entity enables the execution of programs described in the next subsection. The Turing Machine entity's complexity is not included in any of the complexity related measurements.

The program

Valid syntax in the program implementation consists of a set of states, Q, where each state, q, is defined as a set of tuples: {…(value-to-write, tape-direction, next-state)…} such that there is one tuple per alphabet symbol. The alphabet, I, in this emulation is assumed to be a set of integers starting from zero. I maps onto q such that an input value read from the tape points to the correct (value-to-write, tape-direction, next-state) tuple within state q. Note that the value-to-write is written before the tape is moved. The tape-direction (L, R, and S) is represented as (0, 1, and 2). A next-state of negative one (–1) indicates a valid end of program and is included in length and complexity measures of the program. The initial

program has an estimated complexity of 0.7917 and length of 24 bytes.

The input and output tapes for the turing machine

The implementation of the tape, *T*, is simply a string of numbers. An example input tape looks like: (…1,2,3…). The tape is bi-directionally infinite; movement can occur infinitely to the left or the right. Zero (0) is returned when blank spaces are read from the tape. The length and complexity of a tape includes only the values which were initially on the tape or written to the tape during a program execution and not the infinity of zeroes in either direction. In order to prevent explosive growth in the amount of data generated during an execution of the emulation, program and data tapes are implemented as circular queues. Once a specified tape length is reached, the oldest data is overwritten. The initial input data has an estimated complexity of 1.0 and the estimated output data complexity is 0.8571. The initial input length is 5 bytes and output has a length of 7 bytes.

The attractive force

Heat is the attractive force that pulls the three main object types together. The objects generate heat and attempt to maintain an ideal temperature by grouping together. They can also be repelled if the temperature is above the ideal temperature. An accurate model of heat diffusion on the two dimensional grid is included as part of the simulation. Heat generation, evaporation rate, and the diffusion constant within the two-dimensional grid are specified in Table 10. A discrete approximation to diffusion is used from Definition 7.1 where *nbdavg* is the weighted-average of the eight adjacent neighbors. Heat is initially set uniformly for each bug from the range specified in Table 10. The motivation for an entity to move is determined by its unhappiness as defined in Definition 7.2.

This report presents results relating to the search for a fundamental understanding of information assurance. The results from this research enable a deeper understanding that leads towards quantification of vulnerabilities, measurement, control, and composition of information security safeguards, as well as new types of safeguards. The main result of this work is a unified view that enables information assurance to be engineered as an integrated feature of an information system. Results from this work are integrated with an existing tool, called NIPAT, which displays a measure of the security of a system through an electrical engineering paradigm (Section 3). Other attempts have been made to represent information assurance in similar grid formats. NIPAT is used as a representation of generic grid-based techniques so that the strengths and weaknesses of such approaches can be determined.

Table 10 Emulation Control Variables

Emulation Variable	Value
Initial length of data and program strings in bytes	(5, 24)
Initial spatial distribution of strings and programs and turing Machines	Random
Proportion of strings, programs, and Turing Machines	(0.3, 0.3, 0.4)
Functional activity of programs	Shuffle Input Data
Heat evaporation rate and diffusion constant	(0.99, 1.0)
Heat generation of entities chosen uniformly from the range	3,000–10,000
Ideal temperature chosen uniformly from the range	17,000–31,000
Emulation run time in simulation time units	400

Figure 116 illustrates the approach taken in this research towards building a foundation for engineering information assurance. This work began by developing a computational model upon which to test and build information assurance concepts. In the next layer of the information assurance framework a feasible metric was developed through interaction with the information assurance model. In conjunction with the information assurance model and metric space developed through the induced topological framework, insights gained through

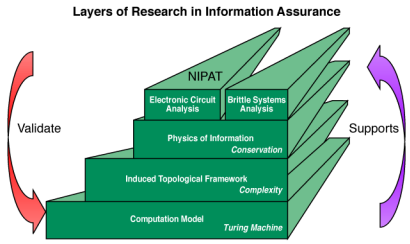

Figure 116. Architectural layers for information assurance.

viewing infor mation as a physical phenomenon were developed. Electrical engineering and brittle systems analysis are specific examples, tangible to users, of these physical insights. While Figure 116 appears as a simple construction with each layer building upon the one beneath it, in reality the research has been an iterative process where one layer is used to help support or validate another layer. As a specific example, the Turing Machine may be viewed as a specific computational model, Kolmogorov Complexity as inducing a metric space, and Brittle Systems, derived from materials science, as an analysis of fundamental tradeoffs in the information assurance performance of a system. Physics of information has contributed towards understanding the conservation of complexity by providing the insight to look for conserved properties related to information assurance. Complexity-based vulnerability analysis (Section 9), sits at the level of the induced topological framework. Complexity-based vulnerability analysis plots the complexity measure of a system. Using this information, the conservation principle from physics of information, brittle systems analysis, or electrical engineering principles may be applied in order to engineer the desired properties into the information system. Learning from the strengths and weaknesses of the above approaches, this research has driven deeper into the nature of information assurance through a series of original hypotheses and theorems. The complexity-based approach derived through this series of hypotheses and theorems provides a general unified theory for understanding the fundamentals of information assurance. The results from our complexity-based approach can be used synergistically with other approaches.

This report begins with a brief overview of relevant research that has attempted to understand the properties of information in terms of well-defined scientific and engineering disciplines. Next, the desirable properties of a metric for security are examined (Section 3.3). In order to further the development of a realistic metric, a general model for studying information assurance is proposed (Section 4). Next, a definition of vulnerability is proposed in terms of the new model based on Turing Machines (Hypothesis 4.1), and engineered properties of information assurance with an analogy to mechanical engineering are proposed in terms of the new model. The analogy with mechanical engineering is called Brittle Systems (Section 5) and involves the design of information assurance in a manner that accounts for tradeoffs in performance and degradation of information assurance in a system. Information assurance is also viewed from the perspective of set theory and a topological space (Section 3.5). This is particularly relevant in understanding the operation of the metric in terms of secure composition and the limits of applying safeguards to a system.

Continuing the outline of the rest of the report, a key contribution of this effort, the development of a particular information metric, is presented. Before this point, the report has examined in detail only the properties of a metric, not an actual metric. The metric proposed is Kolmogorov Complexity (Section 6). The advantages and drawbacks of this metric are discussed, including its incomputable nature. However, computable estimates (Section 6.2) of Kolmogorov Complexity are proposed next, as well as additional useful applications of Kolmogorov Complexity for communications in general. These additional applications are important because they demonstrate how information assurance should be an integral part of information system design. Next Theorems 6.1 and 6.2 concerning the conservation of complexity (Section 6.7) within an information system are discussed. This leads to a Swarm experiment that monitors the evolution of complexity in a dynamic and complex system and examines our ability to monitor the complexity as it evolves. Unless vulnerabilities can be identified and measured, the information assurance of a system can never be properly designed or guaranteed. Results from a study on complexity evolving within an information system using Mathematica (Section 9.2), Swarm, and a new Java complexity probe toolkit (Section 9.4), developed by this project, are presented in this report. An underlying definition of information security is hypothesized (Hypothesis 9.1) based upon the attacker and defender as reasoning entities, capable of learning to outwit one another. This leads to a study of the evolution of complexity in an information system and the effects of the environment upon the evolution of complexity. Understanding the evolution of complexity in a system enables a better understanding of how to measure and quantify the vulnerability of a system. Finally, the design of the Java complexity probes toolkit under construction for automated measurement of information assurance is presented (Section 9.5). Appendix A presents a dialog in which typical

questions about the relationship between complexity and information assurance are posed and answered. This dialog is best read after reading the introduction Kolmogorov Complexity (Section 6.1) or for someone already familiar with complexity theory who wants a quick overview of the approach taken on this project toward the relationship between complexity and information assurance. Appendix B presents the design for an experiment that could be run to validate the complexity-based vulnerability analysis concept (Section 9). Appendix C provides more detail on the design and operation of the NIPAT security tool.

6.2 EXPERIMENTAL RESULTS FROM THE EVOLUTION OF COMPLEXITY

This section presents the results obtained from the emulation and their relation regarding the theoretical definition of complexity discussed earlier in this report as well as their relation to vulnerability analysis and complexity measurement in general. In this emulation, the three types of computationally related entities (data, programs, and Turing Machines) are initially distributed in random locations within the two-dimensional grid-space. Each entity generates a fixed amount of heat during the emulation run. Each entity also has an internal variable termed its "unhappiness" which is a normalized distance of the entity is from its desired temperature. An entity cooler than its ideal temperature will move one grid-space per time unit in the direction of warmth. As entities congregate, heat increases forming red hazy hot spots on the grid.

Begin with a single entity that contains a Turing Machine, program, and data. This single entity will evolve by executing its program upon its current data at each time step. In Figure 117 the number of program transitions is compared and contrasted to the data bit-string length. Note that all bit-strings are contained in circular queues of size 100 bytes. Once the queue is filled, the oldest data is overwritten with newly generated data. Thus, there is a bound on the maximum space available. There are several reasons for using the circular queues. The first is a practical reason; unlimited growth quickly slows the real-time execution of the emulation. Also, in reality, one generally does not feed all available data into a program if it is not necessary for exactly the reason mentioned. Finally, the cumulative effects of new data complexity will more quickly become apparent if the older data is eventually discarded.

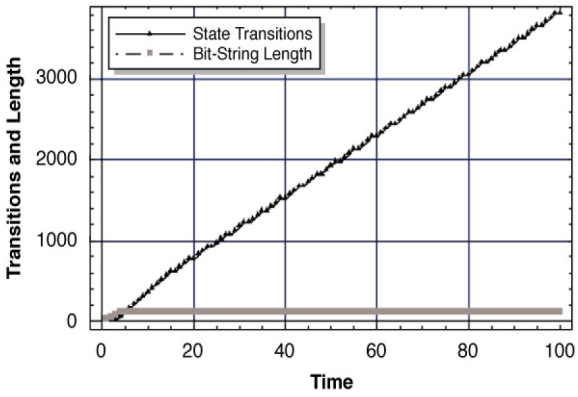

Figure 117. Single Entity Transitions and Length.

In Figure 118, the resulting complexity is plotted

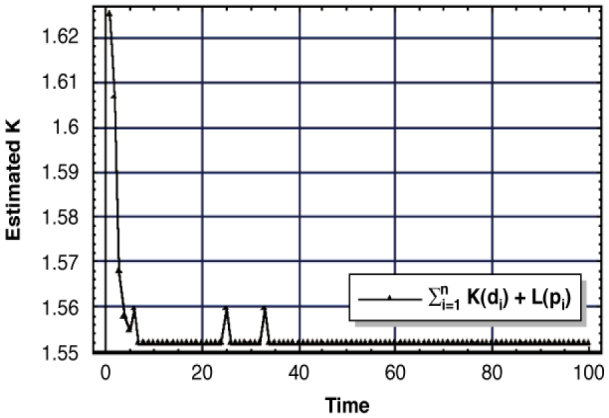

Figure 118. Single Entity Change in Complexity.

for each evolution, that is, for each program execution. Note that this complexity measure includes the program complexity itself. However, as the program does not change for the results presented in this report, the decrease in complexity is due to the data. It should be noted that the program executed shuffles the input data to generate the output data. Clearly, the complexity reaches a minimum constant value in Figure 118.

In order to begin to study the relation of position with complexity, a metric called *envelopment* has been defined as the complement of the inverse of the number of directly adjacent neighbors of an entity as shown in Definition 8.1. The value plotted in the following graphs is the expected value for all entities in the system. High values of *envelopment* cause a reduction in the location complexity. Clustering produces

more heat, thus raising the "happiness" of the entities within the cluster as can be seen in Figure 119.

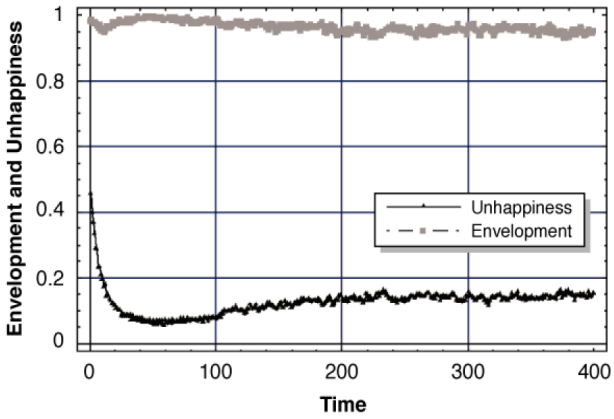

Figure 119. Envelopment and Happiness.

Envelopment or clustering raises the possibility that programs, data, and Turing Machines will meet. When any two different entities meet, the contents of both are copied to each entity. Note that a complete representation of the system would have to include the movement and exchange of data. This is purposely not included in our results; instead, a sample of sub-component complexities is examined. It would be rare, especially in today's communication networks for example, to have complete access to all relationships. Typically only the data points presented by an SNMP MIB, for example, are available. When an entity contains program, data, and Turing Machine, it executes the program generating new data. Figure 120 shows the number of data and program exchanges (labeled *Data Cp, Prog Cp* in Figure 120) with program execution enabled (labeled *w Ex* in Figure 120) and without program execution. Program execution lags data and program exchange by a small amount, because program execution cannot proceed until all three entities meet. Also, when program execution is enabled many more exchanges take place. This is because program execution creates new data to be exchanged. When execution is enabled, the amount

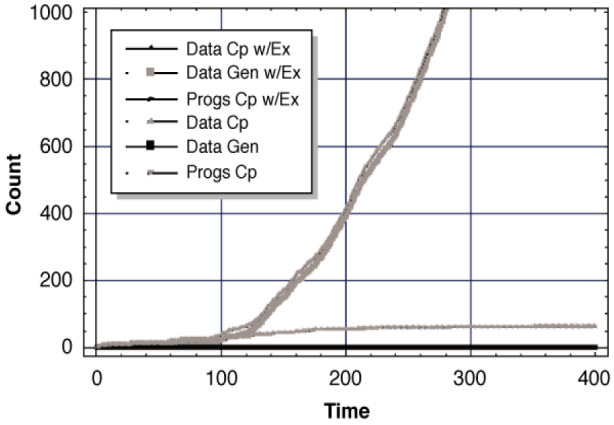

Figure 120. Data Exchanges and Generation with and without Program Execution.

of data in the system rises rapidly. All data previously gathered by an entity, bounded by the size of the circular queue, is fed into a Turing Machine when program execution occurs. Each Turing Machine is equipped with a timeout mechanism so that the program will forcibly end in case of endless loops, or simply too much data. The number of forced-timeouts is shown in Figure 121. Any data generated

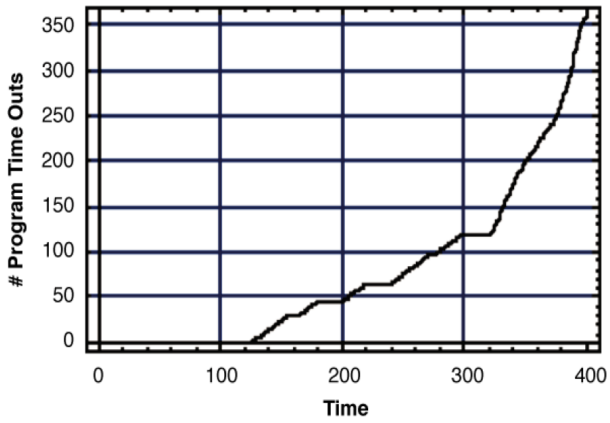

Figure 121. Program Timeouts.

before the timeout is considered valid output. There were no program exceptions in this case.

In Figure 122, the state transition per bit-string

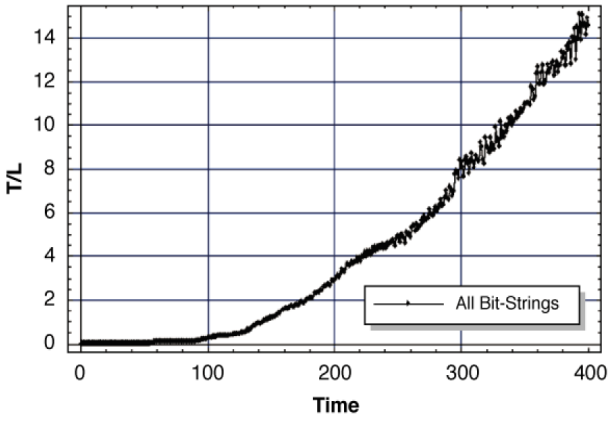

Figure 122. Turing Machine Transitions and Bit-String Length.

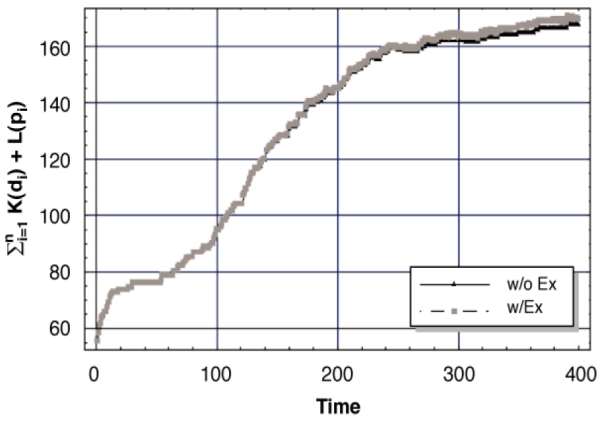

Figure 123. Total System Estimated Complexity.

length shows the total number of Turing Machine State transitions per length of bit-strings. Initially, the system has no program execution, only information exchange. It is possible for a zero, which indicates end of tape, to be shuffled into the middle of an output tape. Only the result up to the zero is regarded as output; the remainder of the program transitions may be considered subjectively as wasted effort. Thus, the Turing Machine programs can execute state transitions without generating the equivalent amount of output.

Now consider the focus of this work, complexity. The complexity estimation used in this analysis is the compression-based estimate of complexity described earlier. In this emulation, the three main types of entities are tagged and certain rules are enforced. Thus, for example, data cannot execute a program and a program cannot attempt to execute a Turing Machine. Given this case, the authors hypothesize that a given bound on the complexity will eventually be reached in the system. In Figure 123, the estimated complexity of the system grows faster when program execution is enabled as contrasted with entity exchange only. This is a key result. In a closed environment with a fixed number of programs and data, the only way for the complexity of an entity to change is through exchange of information. No new information is created except through copying information from one entity to another. When the Turing Machines are enabled in each entity, they shuffle the input data, creating new data that is then available to be exchanged with other entities.

Compare and contrast the total bit-string length with the total bit-string complexity throughout the emulation shown in Figure 124 with program execution. Notice that the total system bit-string length increases at a much greater rate than the complexity. This occurs because the initial activity is data and program exchange as entities meet within the system and data spreads through sharing. As information sharing decreases because each entity already has a copy of the information, the primary activity becomes data generation through program execution.

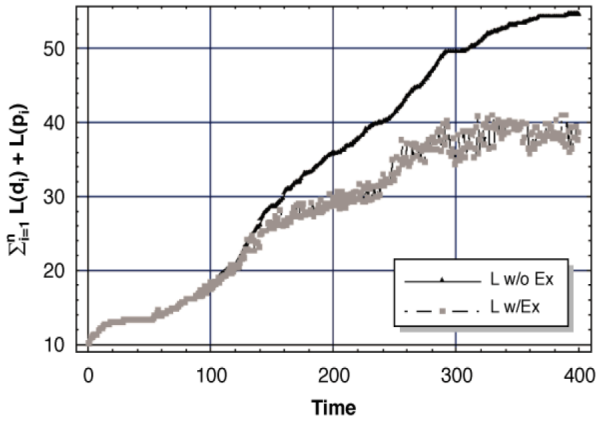

Figure 124. Bit-String Length.

In Figure 125, the complexity per length of data is plotted. Notice that with program execution enabled the complexity per length of data increases. In the case of operation without program execution, only a finite length of data exists. When the total estimated complexity of the data within each entity is plotted over time, the system with program execu-

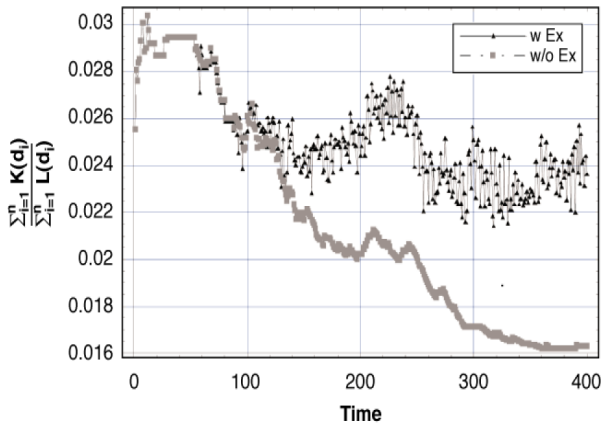

Figure 125. Complexity per Length.

tion shows a marked increase in estimated complexity over the pure exchange system.

In Figure 126, the expected value of the complex-

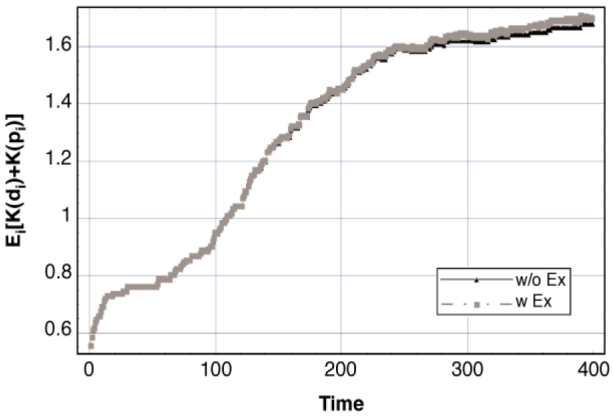

Figure 126. Expected Complexity over Time.

ity of all bit-strings contained by an entity is plotted versus time. The larger curve shows the average complexity per entity when program execution is enabled. As expected, program execution with this particular program, one that shuffles data, increased the average system complexity. Note that the difference in complexity does not appear until approximately time 100. However, program execution events began slightly before time 100. A question important for systems that would make decisions based upon complexity concerns the delay between the time of the cause of complexity change and the time for a measurable increase in complexity to occur. This question is an interesting one that is outside the scope of this report.

This emulation validated several intuitive concepts. The first involves complexity in a closed system; that is, a system defined as a fixed number of entities in which no data, programs, or Turing Machines may enter from outside the system. A program injected into the system that, on a microscopic scale increases complexity generated greater average macroscopic system complexity. The results of this study also help to validate our initial hypothesis: the greater the complexity, the more effort required to understand it. The emulation with program execution, showing a higher complexity, required many more probes in order to understand its behavior than the system with only program and data exchange enabled. This implies that an attacker, with no *a priori* knowledge of the system, would have a more difficult time understanding how to understand and thwart the more complex system. This leads to the next section on complexity-based vulnerability analysis.

6.3 COMPLEXITY-BASED VULNERABILITY ANALYSIS

Automated Discovery of Vulnerabilities without a priori Knowledge of Vulnerability Types

Any vulnerability analysis technique for Information Assurance must account for the innovation of an attacker. Such a metric was suggested about 700 years ago by William of Occam [27]. Occam's Razor has been the basis of much of this report and the complexity-based vulnerability method to be presented. The salient point of Occam's Razor and complexity-based vulnerability analysis is that the better one understands a phenomenon, the more concisely the phenomenon can be described. This is the essence of the goal of science: to develop theories that require a minimal amount of random information. Ideally, all the knowledge required to describe a phenomenon should be algorithmically contained in formulae. Observe an input sequence at the bit-level and concatenate with an output sequence at the bit-level. This input/output concatenation is for either the entire system or for components of the system. If there is low complexity in the input/output observations, then it is likely to be easy for an attacker to understand the system. Hypothesis 9.1 explicitly states the means of measuring the com-

plexity of a system component, or protocol interaction, to a potential attacker.

Mozart and vulnerability analysis

Science is art and art is science. One of the most mathematical of art forms is the composition of music. Music is compressed and transported over the Internet very frequently, and most listeners of such music probably have little interest in the compression ratio of a particular piece of music. However, this piece of information can be very interesting and informative with regard to the complexity of a piece of music. One would expect an incompressible piece of music to be highly complex, perhaps bordering on random noise, while a highly compressible piece of music would have a very simple repetitive nature. Most people would probably prefer music that falls in a mid-level of complexity; sounds that are not repetitious and boring yet not random and annoying, but follow an internal pattern in the listeners' minds. Music is a mathematical sequence that the composer is posing to the listener; the more easily the listener can extrapolate the sequence without being too challenging or too easy, the more pleasing the music sounds. Carrying the music analogy forward in a more explicit manner, consider the listener as an attacker and the composer as the designer of an information system. If a user has a preference for a given type of music, a sample of that music can be included as a hypothesis in an MML-based complexity analysis. The lower the complexity the more appealing the music to that particular listener. The more easily the listener can extrapolate the musical sequence, the more vulnerable the system.

Imagine the composer who wishes his music to be enjoyed by only a specific group of listeners and no others. The composer is constrained from generating a completely invulnerable system, that is, totally random, because the composer wants the music to be meaningful to at least some potential group of listeners. Relating this analogy to the hydrostatic test that was mentioned in the introduction to this report, vulnerability is the quantification of the potential leakage of music that is enjoyable to unintended listeners.

In a very quick experiment, the following three pieces of music were tested for complexity: Beethoven's Sonata Op. 27, No. 2 ("Moonlight"), Mozart's Sonata in A Major ("Alla Turka"), and Philip Glass's Opening to "Glassworks." The encoding explicitly represents notes, timing, dynamics, and phrasing. Beethoven's Moonlight Sonata has a complexity rating of 0.13, Mozart's Sonata has a complexity rating of 0.16, and Glass's Introduction to Glassworks has a complexity of 0.03. Philip Glass makes extreme use of repetitious arpeggios in his work, thus the low complexity rating. This author expected Beethoven to have a slightly higher complexity than Mozart by a small amount, but it was the reverse in this case. Note that one could decompose the overall complexity to determine the complexity of a composer's use of rhythm, note structure, phrasing, or other musical components. Once a composer's typical complexity band is benchmarked; this type of analysis could be used as an indicator to determine authenticity. Additionally, one could conjecture that "learning" to model Beethoven or Mozart might be more difficult than other composers.

Experimental validation of complexity-based vulnerability analysis

A model information system has been implemented in Mathematica to begin experimental validation of complexity-based vulnerability analysis. The goal is to determine the vulnerability, not only of the overall system, but also of system components. Vulnerability analysis should be done without any *a priori* knowledge about system operation or knowledge of particular types of vulnerabilities. Expert systems and vulnerability analysis tools that rely upon rules identifying particular types of vulnerabilities are inherently brittle and, in fact, meaningless against an innovative attacker. Our Mathematica information system model purposely does **not** include component descriptions or explanations because the goal is for the system to be a black box with respect to vulnerability. The point is that a vulnerability analysis can be done without having to know the details of the system. At the end of this analysis the functions of the analyzed components are mentioned. It should then make intuitive sense that a particular component performing a simple operation had a lower complexity than one performing a more "random" operation.

Each component of an information system modeled in Mathematica contains probe points through which bit level input and output can be collected. A complexity function based upon a simple inverse compression ratio is used as an estimate of complexity. The intent is to experiment with better complex-

ity measures as the project continues.
Figure 127shows results from complexity measures taken of accumulated input and output of three separate components of the toy information system. The graphs show the complexity of bit-level input and output strings concatenated together. That is, observe an input sequence at the bit-level and concatenate with an output sequence at the bit-level. This input/output concatenation is for either the entire system or for components of the system. If there is a low complexity in the input/output observations, then it is likely to be easy for an attacker to understand the system, as in Hypothesis 9.1. Note that these graphs are showing estimates of Kolmogorov Complexity. If MML [37] were used, the attacker's hypothesis would be used to determine the complexity relative to a particular attacker. In Figure 127 the X-axis is the number input and output observations concatenated to form a single string of bits. The particular complexity estimate used in this example is very poor; however, an ongoing area of research is to improve complexity estimates. Because of the inaccurate complexity metric, all the figures show a rising complexity with the number of accumulated observations. However, notice the rate at which the complexity rises in each of the figures. From Table 11, it would appear that Component E is most vulnerable due to its low rate of increase in complexity while Component B appears to be the least vulnerable due to its steeper rise in complexity. These results make intuitive sense because Component E is simply transmitting data without any form of protection while Component B is adding noise to the data. This vulnerability method does not take into whether a component reduced or increased complexity; in other words whether the change was endothermic or exothermic complexity behavior.

Table 11 Component Vulnerabilities

Component	$K(x[op_{start}:]op_{end})$
C	19.6011
B	19.6302
E	19.0013

These results show that vulnerabilities can be systemically discovered. These vulnerabilities can be quantified to a value within the bounds of the complexity measure error. When used in an MML [37] approach to complexity measurement, apparent complexity, that is, complexity as seen by a particular attacker can be determined. Thus, this work has led towards automatic generation of vulnerabilities without requiring expert knowledge of each type of vulnerability and in a more complete manner, depending upon the number of components analyzed. Note that all possible combination of components must be analyzed in this manner.

An information system should be designed in such a manner that the apparent complexity of the system under attack can be determined with respect to the attacker and that information used to maximize the distance in the apparent complexity between the attacker and defenders in an automatically reconstituted system. An Active Network [16] is an ideal environment in which to experiment with an implementation of automated system reconstitution because it provides extreme flexibility in fine-grain code movement and composition of code. Apparent Complexity is used to reconstitute the system such that the complexity difference is maximized between legitimate users and attackers of the system. In this section, the discussion is limited to the automated hardening of a system based upon information about an attacker and a new form of vulnerability analysis.

Design of a complexity-based vulnerability analysis tool

A vulnerability analysis tool should quickly and efficiently identify and display the vulnerability of an information system ranging from an application, to a node, network, or an interconnection of networks. The tool should be portable, easy to use, and have minimal impact upon a system. The tool should also be integrated with the management of the system or network. The approach taken has been to use the lessons learned from the emulation in this analysis to consider the design of a complexity-based vulnerability analysis system within the context of network management. The Swarm framework in the emulation previously described queries potentially large numbers of entities much like SNMP polls data for system management. A salient difference is that in a Swarm model, simulation time can be controlled in order to make certain that all data is queried at precisely the requested simulation step interval. However, design and minimization of the number of data points to be collected, such that system operation can be described as fully as possible, is exactly the

same in actual Management Information Base design for system and network management.

While integration with system management using a de facto standard such as SNMP is a future goal, a set of Java packages implementing lightweight "probes" that transparently gather data from a bit stream and report the result to the vulnerability analysis portion of the Java package has been developed. This design was chosen because the implemented probes are extremely lightweight and also initiate the transfer of bit-level data, rather than waiting to respond to a management query. Once a better understanding of complexity is obtained, transition to implementation in SNMP will likely take place. The Swarm emulation ran with 200 such probes, one in each entity object and successfully reported the results to the analysis package. The authors intend to pursue the study of complexity within an active network environment [16], particularly with regard to network reconstitution in the face of attack.

Future work to be addressed includes the best way to present data collected from hundreds of complexity probes and how to incorporate that data into a useful integrated view, particularly in a network management context. Much more detailed analysis of the complexity emulation results is required as well as modifications to the basic emulation in order to understand the effects of access controls in the exchange of information, the role of complexity in automated service composition and fault tolerance, and how complexity can best be used for determining network attacks and fault conditions.

Applying complexity to vulnerability analysis

Vulnerability analysis is defined as the process of quantifying the vulnerability of an information system to attack. An attack is defined as the act of an unauthorized user extracting unauthorized information from a system. As discussed previously, the information system appears to the attacker as a natural physical entity to be researched; its behavior explored. Complexity, in the colloquial sense, should be high for the attacker and low for legitimate users. Kolmogorov Complexity is an omniscient being's measure of absolute complexity; by definition it measures the size of the smallest program that can be generated. In a later section apparent complexity is introduced as a potentially more feasible measure. However, the use of complexity for information assurance in general is discussed and the term complexity will refer to Kolmogorov Complexity in this section. In the steps that follow, a simple, idealized approach is discussed. Many complicating details need to be addressed if this is to be literally applied, however, it provides a starting point for exposition of the concept.

The first step in computing complexity is to map the entire observable portion of the information system to a string. This string ideally represents data points collected over the entire system at every instant in time. Note that a data system includes arbitrary input, computation, and output. This is included as part of the string. Including every possible input and output will result in an extremely long string. From Definition 6.1, complexity is the length of the minimal program running on a Universal Turing Machine that is capable of generating that string. There is only one possible length for the shortest program; thus the result is a single, ultimate quantification of complexity.

The ability of the attacker to compute the complexity of portions of the above string is directly relevant to the attacker's understanding and ability to predict future behavior of the system. This includes understanding the system's vulnerabilities. Using the method of computing complexity as described in the previous paragraph would lead to an infinite string; it is necessary to map the infinite into the finite in order to make the process feasible.

There are a few observations that make the process more feasible. The first is results-based scoping. For example, the attacker is most likely to be interested in certain very specific results, such as obtaining specific types of information. Thus, the attacker can attempt to narrow the observation points in the string to only those appear to be promising. On the other hand, the defender of the information system is primary concerned with intrusions and other fault behavior that compromises Information Assurance. Thus, the defender can narrow the scope to strings that contain results that lead to those outcomes.

The second is spatial scoping of the string. For example, the defender can compute the complexity of various components in the system, instead of the entire system. For example, only portions of the system relevant to information access can be analyzed. The result is a mosaic of localized complexity measures of the system. This is equivalent to computing complexity over various portions and widths of the string. As shown in Section 6.2, the composition of

two components results in a complexity that is no larger than the sum of the complexities.

The network insecurity path analysis tool and complexity-based vulnerability analysis

An experimental prototype tool has been developed that combines the grid-based vulnerability analysis technique with the complexity-based vulnerability analysis method developed in this project. This section discusses the enhanced tool shown in Figure 127.

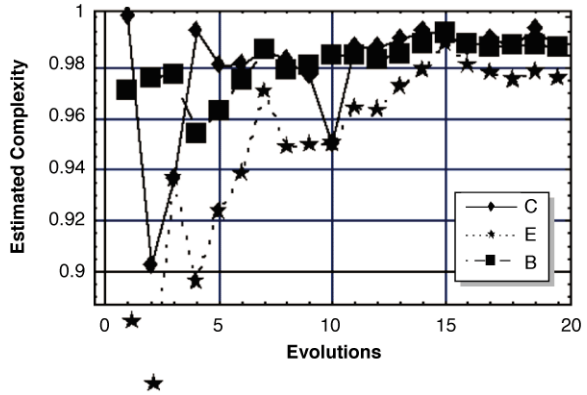

Figure 127. Prototype tool combining the grid-based vulnerability analysis technique with the complexity-based vulnerability analysis method.

Every information system is assumed to take data of some form as input, process the data, and return data as output in some form. Essentially, every information system can be defined as a mathematical operation. Information systems developed by humans today tend to be highly structured in order to be tractable in their development and maintenance. Generally, there are well-defined data flows and processing functions within the information system. The system is composed of a hierarchical composition of functional units. This is especially true in legacy network systems where layered design is ubiquitous. For these systems, one can imagine complexity probes located at the input and output of every functional unit in the system. This allows determination of the vulnerability of each process and data stream at a very granular level. This provides a complexity-based vulnerability map for the system. A potential attacker would be unlikely to have such a detailed understanding of a target information system; an optimization to this technique is to limit probe locations to only those locations likely to be observable to an attacker. The vulnerability map is used to determine insecurity flow through the system. Complexity is viewed as resistance to attack. Both a most likely path and a maximum flow algorithm are applied in this experimental complexity-based vulnerability analysis tool. The most likely path is determined by finding the lowest complexity path from a given attack point to a given target point. The maximum flow algorithm assumes that lower complexity paths have a greater capacity. The question arises as to what "flow" means in terms of complexity. Firstly, the entire foundation of complexity-based vulnerability analysis rests upon the likelihood, or probability, of attack being successful upon the low complexity locations of an information system as per Hypothesis 9.1. The complexity probe values are displayed as links in the complexity tool display shown in Figure 127. The values of the links are $1/K$ and these values are normalized to 1.0 for each node in order to obtain a probability of successful attack upon each link. The maximum flow algorithm provided by this tool shows the optimized placement of resources by an attacker to maximize the likelihood of a successful attack.

The distribution of insecurity information

The approach to measuring the complexity of a system, as described throughout this document, results in determining the ease with which a potential attacker can understand the system. It does not directly account for the fact that information about the target system can be obtained by a potential attacker in algorithmic form, that is, in the form of an attack tool. Such a tool does not require the attacker to understand its operation. The attack tool is like an active packet, or a parasite that depends upon its host for transportation. This is distinct from a virus, whose primary function is replication and transport. For example, an attacker may have little understanding of a particular system, yet download an attack tool that allows the attacker to perform a successful attack. Thus, the distribution of attack knowledge needs to be considered. Once an attack tool is in the hands of an attacker, the apparent complexity is greatly reduced. There is an interesting feedback mechanism here; data that can reduce the apparent complexity to a potential attacker needs to be kept secure by the defender. Once obtained by an attacker, a significant drop in apparent complexity occurs, potentially leading to further significant reduction in apparent complexity as more vulnerability information is obtained and disseminated to other attackers.

One might view the evolution of complexity in the following terms. An information system is built. Initially, an attacker discovers its least complex components. The attacker decides to automate his attack (active) and/or publish the mechanism to accomplish the attack (passive). This information is disseminated through the population. Meanwhile the information system defenders, usually after considerable delay, discover the attack mechanism and patch the hole. The population of attackers, building upon their knowledge, exploits the next least complex link from their view in the information. The defenders eventually close this hole. The cycle continues *ad infinitum*. The cycle of attack and defense can be viewed through complexity as a cycle, or evolution of complexity as shown in Figure 128. Low complexity portions of a system will eventually be learned and disseminated by an attacker. To account for this dissemination of low complexity information, defenders reinforce the low complexity areas with more complexity. The results of this project allow system developers to understand not only where the vulnerable portions of the system are located, but to engineer their systems in such a manner as to control the cycle shown in Figure 128. This process can be modeled as low complexity portions of an information system that evolve in complexity over time.

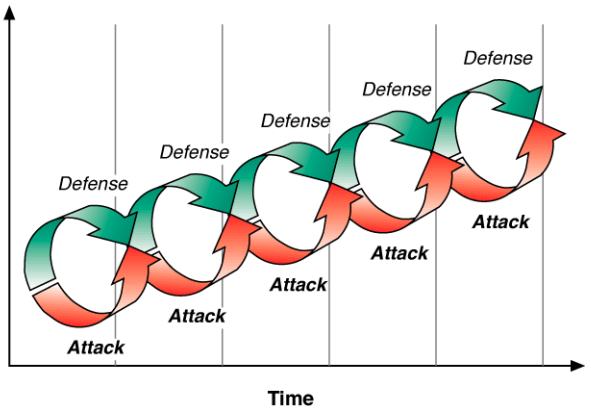

Figure 128. Cycle of attack and defense viewed through complexity as a cycle, or evolution, of complexity.

7. Self-Healing Information System

A Coherent, Self-Healing System Self-Composition, Genetic Algorithms, and Kolmogorov Complexity

Fault tolerant and self-healing systems should have the ability to self-compose solutions to faults. This should be an inherent part of system operation, rather than a structure imposed from "outside" the system. Genetic Algorithms are on the path towards self-composing solutions, however genetic algorithms, as implemented today, require external control to manipulate the genetic material. In other words, the genetic algorithm itself must be programmed into the system; if the genetic algorithm code failed, then the self-healing capability would fail. While this situation is not ideal, it is explored as a possible step towards a truly self-healing system.

Active networking is a novel approach to network architecture in which network nodes—switches, routers, hubs, bridges, gateways etc. —perform customized computation on packets flowing through them. The network is called an "active network" because new computations are injected into the nodes dynamically, thereby altering the behavior of the network. Packets in an active network can carry fragments of program code in addition to data. Customized computation is embedded within the packet's code, which is executed on the intermediate network nodes.

Many active network components and services have been designed, implemented, and are undergoing experimentation. The ABone (Active Network Backbone) implements a relatively large-scale (given the novelty of the technology) active network (*O(100)* nodes). However, the fundamental science required to understand and take full advantage of active networking is lagging behind the ability to engineer and build such networks. In fact, the current Internet, whose protocols were built upon the ill-defined goal of simplicity are only slowly being understood. An outcry from the Internet community, with its carefully crafted, static protocol processing, with massive documentation *(O(4000)* Request for Comments) of passive (non-executable) packets is that it is already "too" complex.

An adaptive fault tolerant system, no matter how resilient, would unlikely receive acceptance by industry or the community if it were considered "complex" in the colloquial sense. How can such systems, which require complexity to be adaptive, at the same time appear simple to understand and manage. Are active networks really more complex than the current Internet? Are adaptive applications built upon active networks any more or less complex than the same applications built upon the legacy Internet? Does a measure of complexity exist that would allow an objective comparison to be made? What are the benefits of an active network with respect to passive networks? While these are extremely difficult questions to answer, this report attempts to lay the groundwork for answering these questions by proposing a complexity measure, Kolmogorov Complexity, and proposing an adaptation mechanism, Genetic Programming, based upon an analogy with biological systems.

Kolmogorov Complexity was applied as a measure of potential algorithmic information content for use in prediction and control of an active network [185]. In the remainder of this paper, the term complexity will be used to indicate a particular form of complexity known as Kolmogorov Complexity. Kolmogorov Complexity is a measure of the length of the smallest program, such that, when executed upon a Universal Turing Machine, it generates a particular string of bits *x*. The length of such a smallest program *K(x)* is the complexity of the bit-string, *x*. It should be noted that research has been performed in the use of genetic programming to evolve the smallest program for a given bit-string, and thus estimate *K(x)*. Complexity was applied to optimize the combined use of communication and computation within an active network; to determine the optimal amount of code versus data [185]. It was shown that if the Kolmogorov Complexity of the information related to the prediction of the future state of the network is estimated to be high, then the ability to develop code, representing the non-random, or

algorithmic portion, of that information is low. This results in a low potential benefit for algorithmic coding of the information; the benefit of having code within an active packet would appear to be minimal in such cases. Conversely, if the complexity estimate is low, then there is great potential benefit in representing information in algorithmic form within an active packet. It was suggested that if the algorithmic portion of information changes often and impacts the operation of network devices then active networking provides the best framework for implementing solutions[185]. This is precisely the case in genetically programmed network services, a new class of services that are not pre-defined but those that evolve themselves in the network in response to the state of the network. In this report, we will restrict this class to those services that are programmatic solutions for perceived faults that occur in a network. Further research is required to generalize this class to include other types of network services.

Frameworks for protocol and service composition have been developed for active networks, one of which is well described [186]. Thoughts on the requirements for protocol and service composition are also discussed in [187]. However, the work done to date is lacking in that it does not address how active code will be generated rapidly enough to make dynamic injection of the code a significant factor. The argument against active and programmable networks is that, given enough time, memory, and processing power, legacy systems could eventually contain all the functionality that active networks could have injected. To do this, legacy developers would have to know {\em a priori} all possible functionality that would be required in the network. However, this report demonstrates that it is possible for the network to generate code rapidly and in a manner that can never be known {\em a priori} for every possible condition. The inspiration for a genetic algorithm based approach to solution composition comes from nature in the form of the docking problem in molecular biology [188]. Solutions that efficiently match a particular fault should be able to “dock” with the fault. Prediction for successful docking in biology can be attempted by searching for minimal energy or minimal geometric construction combinations. Here we consider a genetic algorithm used to generate a solution for the self-composition of solutions to mitigate network faults. One goal of the experiment discussed later in this report is to study the relationship between complexity and solution composition. In particular, it has been hypothesized that the complexity of the fault and potential solution will decrease as the optimal solution is composed. Specific examples of faults that could be simulated are:

- Network mis-configuration
- Bandwidth and Processor mis-allocation
- Faults caused by Distributed Denial of Service and virus attacks
- Poor Traffic shaping
- Routing problems
- Non-optimal fused data within the network
- Poor link quality in wireless and mobile environments
- Mal-composed protocol framework models in the network
- Poorly tuned components of network services

A simple fault, namely, mis-allocation of bandwidth and processing capability resulting in packet jitter, has been chosen as a working example. A fitness function defines a metric for “goodness” of a population. In this case, “goodness” is the reduction in the variance of packet arrival times. The fault is represented by the difference between the actual system and a minimum required fitness. Genetic material will evolve to minimize the effect of the fault. The complexity of the combined fault-solution pair should be at a minimum when the fitness is optimal. We will borrow a term from molecular biology and call a perfectly matched fault and solution a successful “docking”.

COMPLEXITY AND EVOLUTIONARY CONTROL

Complexity and evolution are intimately linked. Kolmogorov Complexity *(K(x))* is the optimal compression of string *x*. This incomputable, yet fundamental property of information has vast implications in a wide range of applications including system management and optimization[192] [193], security [194],[195], and Bioinformatics. Active networks [196] form an ideal environment in which to study the effects of trade-offs in algorithmic and static information representation because an active packet is concerned with the efficient transport of both code and data. As noted inFigure 129, there is a striking similarity between an active packet and DNA. Both carry information having algorithmic and non-algorithmic portions. The algorithmic portion of DNA has transcription control elements as well as the codons [197]. The active packet has control code and may contain data as well.

Kolmogorov Complexity and Genetic Programming have complementary roles. Genetic Programming has been used to estimate Kolmogorov Complexity [198], [199]. Genetic Programming benefits from Kolmogorov Complexity as a measure and means of controlling not only the complexity, but the size and generality of the result [200]. One of the most obvious uses for complexity in networking is Programmatic Compression [201]. In this report, the foundation is developed for the use of complexity to enable the network to self-heal. In the next section, a description of the Minimum Description Length algorithm and its role in Active Networks is explained.

THE APPLICATION OF COMPLEXITY IN A COMMUNICATIONS NETWORK

The goal of the system that has been implemented is to utilize the benefit of an active network to automatically generate solutions that bring the network back into line with a healthy model of the system. The fitness function is used to describe the desired outcome. The concept of molecular docking, mentioned previously, requires a more precise measurement of the degree of "fit" in the docking of a fault and solution. In this project, we are exploring the use of Kolmogorov Complexity, estimated via the Minimum Description Length algorithm, as the means to measure the fit between the fault and the desired state. The next paragraph describes the Minimum Description Length complexity estimator and its relationship to active networking.

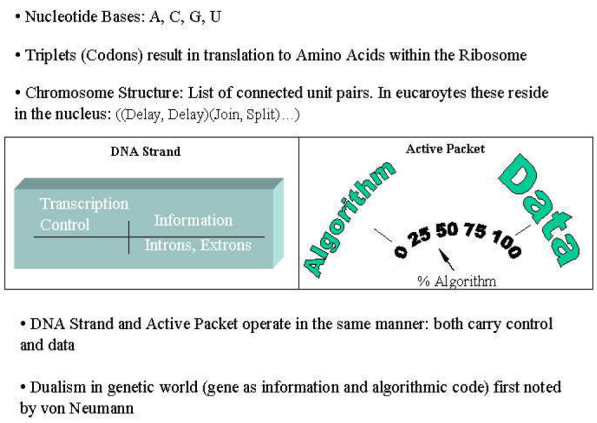

Figure 129. DNA and an Active Packet.

A question active network application developers must answer is: "How can I best leverage the capabilities that active networks have to offer?" Because the word "active" in active networks refers to the ability to dynamically move code and modify execution of components deep within the network, this typically leads to another question: "What is the optimal proportion of content for an active application that should be code versus data?" A method for obtaining the answer to this question comes from direct application of Minimum Description Length (MDL) [202] to an active packet. Let D_x be a binary string representing x. Let H_x be a hypothesis or model, in algorithmic form, that attempts to explain how x is formed. Later in this report, we view H_x as a predictor of x in the analysis of Active Virtual Network Management Prediction. For now let us focus on developing a measure of the complexity of x. MDL states that the sum of the length of the shortest encoding of a hypothesis of two components will estimate the Kolmogorov Complexity. The two components are the length of a model generating string x and the length of the shortest encoding of x using the hypothesis. This can be represented mathematically as $K(x) = K(H_x) + K(D_x|H_x)$. Note that error in the hypothesis or model must be compensated within the encoding. A small hypothesis with a large amount of error does not yield the smallest encoding, nor does an excessively large hypothesis with little or with no error. A method for determining $K_{(x)}$ can be viewed as separating randomness from non-randomness in x by "squeezing out" non-randomness, which is computable, and representing the non-randomness algorithmically. The random part of the string, that is, the part remaining after all pattern has been removed, represents pure randomness, unpredictability, or simply, error. Thus, the goal is to minimize $l(H_e) + l(D_x|H_e) + l(E)$ where $l(x)$ is the length of string x, H_e is the estimated hypothesis used to encode the string (D_x) and E is the error in the hypothesis. The more accurately the hypothesis describes string x and the shorter the hypothesis, the shorter the encoding of the string. Choosing an optimal proportion of code and data minimizes the packet length.

The proposed hypothesis is that the Kolmogorov Complexity of a combined fault and solution description is minimized when the optimal solution to mitigate the fault is composed. A nearly trivial example can be seen with reverse code. Assume that fault data, F exists. Assume that the fault does not erase data but merely transforms it. Define the algorithmic description of the fault data $P_F()$. The reverse code for $P_F()$ will be labeled $RP_F()$. Assume

$P_F()$ and $RP_F()$ are minimal length programs. Then, $RP_F(P_F()) = \phi$, where ϕ is the empty set. RF is the data generated by $RP_F()$. Since the fault does not erase any data, the process is reversible [191] and therefore, $K(RF) - K(F) = 0$. The equivalence in complexity of RF and F follows because there is no loss or gain of complexity when the system is restored to a prior state using the anti-fault process RP_F; there is no work performed. The algorithmically reversed fault will be referred to as an anti-fault in this report.

The descriptive complexity of the fault and the solution should ultimately be as low as possible and the Minimum Descriptive Length algorithm can be used, among other complexity estimators, as a technique to guide solution composition. In fact, this is the case with reversible code. Complexity is important information because it is an indicator of both the type of fault and level of difficulty in correcting the fault and the severity of the fault; fault severity is important in triage operations to optimize system health. Second, a more compact algorithmic representation of a fault will travel faster and more rapidly through the network; it is an efficient format for alerting system management and in triggering automated solutions. Third, it can be relatively easy to reverse the code of an algorithm, possibly generating an anti-fault, or solution to a problem in certain cases. Reversible code has been presented in previous work as a mechanism for generating anti-messages in Time Warp simulation [203].

Fault tolerant and self-healing systems should have the ability to self-compose solutions to faults. Ideally, composition should be an inherent part of system operation, rather than a structure imposed from "outside" the system. Genetic Algorithms are on the path towards self-composing solutions, however genetic algorithms, as implemented today, require external control to manipulate the genetic material. In other words, the genetic algorithm itself must be programmed into the system; if the genetic algorithm code failed, then the self-healing capability would fail. While this situation is not ideal, it is explored as a possible step towards a truly self-healing system.

One of the contributions of this report is the study of complexity in genetic algorithms with the goal of eventually designing self-composing solutions. Genetic algorithms are widely known for their ability to find optimal solutions, avoiding local extremes, by using evolutionary-like processes dependent upon "random" mutation. Kolmogorov Complexity describes the randomness of information. The Kolmogorov Complexity of the genetic material during the evolution of a genetic algorithm can be estimated and yields interesting clues about the underlying physics of the information during its evolution towards a fitness function. It is our hypothesis that, as the evolution proceeds and the fitness level of the genetic material rises, the complexity decreases. This result yields an interesting insight that supports the hypothesis that "solutions" that self-compose to mitigate a fault will tend to decrease in complexity.

THE GENETIC ALGORITHM

The goal of this study is to examine how complexity, specifically an estimate of Kolmogorov Complexity, relates to the evolution of a self-composing solution. We consider a genetic algorithm to be an approximation of a self-composing system. Details on the operation of genetic algorithms can be found in [204] [205] [206]. This paper assumes a basic understanding of genetic algorithm operation and provides only a brief overview. In this experiment a pre-existing Mathematica genetic algorithm package[3] is used. The decision to use Mathematica was based upon its combination of symbolic and arithmetic capabilities and because many of our research utilities, including Kolmogorov estimation functions are implemented in Mathematica.

The genetic algorithm package assumes a population of binary strings of preset size and whose values, when converted to a float type, are between zero and one. Similarly, the fitness function is assumed to accept and return values in the range from zero to one. Fitness values closer to one are assumed indicate more highly optimized results. A genetic algorithm consists essentially of three parts: selection, crossover, and mutation. In selection, each string is selected with a probability proportional to its fitness value. In crossover, a pair of selected strings is determined, a position along the string is chosen at random, and the right and left parts of each string are swapped. In mutation, each gene is changed at random with a low probability, in this case a probability

3. Written by Mats G. Bengtsson National Defense Research Establishment Box 1165, S-581 11 Linkoping Sweden email: matben@lin.foa.se

of 0.002 was chosen based upon repeated experimentation. Each individual is coded as a binary string of length 10 bits. This length provides the size necessary to achieve numerical precision while being small enough to allow a large population size and without excessive overhead. The problem is limited to one-dimension with value x, which represents the real value of the bits in string x, that varies from zero to one. The first step is to create a random population. The population is defined on the real axis from zero to one. The random values are represented in the form of binary strings. Next a fitness function is defined. It is defined in the interval zero to one. The fitness function in this example is defined as $fx = \sin(\pi x)$. Thus, binary representations of values that are odd multiples of 0.5 will have maximal fitness.

Kolmogorov Complexity

This section discusses a general approach for self-composing solutions using lessons learned from the previous section. The approach can be described as the automated generation of a solution hypothesis $H_s = R(H_e - H_f)$, that is, the reverse of the algorithmic difference between the faulty and correct algorithmic representation of behavior by controlled means. As H_f deviates from H_e, complexity or heat as presented here, is generated. In [192] the relationship between fault and energy is explored and simulated (see [195], [194], [207] for recent work on complexity and energy and Information Assurance). The motivation for that experiment came from the relationship between Kolmogorov Complexity and entropy. The definition and application of Kolmogorov Complexity to vulnerability analysis identified how Kolmogorov Complexity can be used to determine vulnerabilities in a system as areas of low complexity. An underlying hypothesis of our work is that computation and communication are fundamentally related through complexity theory, and, thus, bandwidth and processing utilized in denial of service are fundamentally interrelated. Low complexity data or code consuming large amounts of bandwidth or processing indicates the likelihood of an attack. A model of complexity evolution within a closed system is described in reference [194]. That reference developed an abstract model with which to study complexity, specifically Kolmogorov Complexity, of information within an information system. That model explores $K(x)$, a measurement of length in bytes, and $K(x)/s$, a measure of the maximum increase in complexity of the system due to code entering a system such as code carried by active packets. The rate of complexity increase in terms of algorithmic active packet complexity in units of $K(x)/s$ within the closed system was measured. Significant changes in system complexity indicate the presence of faults. Reference [208] reported the results of Kolmogorov Complexity probes that detect Distributed Denial of Service attacks.

An active network environment is used to emphasize that information assurance laws must be able to deal with many alternative and dynamically changing representations of information. With regard to active packets and information theory, passive data is simple Shannon compressed data, and active packets are a combination of data and program code whose efficiency can be estimated by means of Kolmogorov Complexity [209]. The active network Kolmogorov Complexity estimator is currently implemented with a variety of compression estimators ranging from simple empirical entropy to more complex algorithms beyond the scope of this conference. The probe returns an estimate of the smallest compressed size of a string. The simplest estimator, trading accuracy for speed and low overhead, is based upon computing the entropy of the weight of ones in a string. Specifically it is defined in Equation 19where $x\#1$ is the number of 1 bits and $x\#0$ is the number of 0 bits in the string whose complexity is to be determined. Entropy is defined in Equation 20. See [209] for other measures of empirical entropy and their relationship to Kolmogorov Complexity. The expected complexity is asymptotically related to entropy as shown in Equation 21. Observe an input sequence at the bit-level and concatenate with an output sequence at the bit-level. This input/output concatenation is observed for either the entire system or for components of the system. Low complexity input/output observations quantify the ease of understanding by a potential attacker. Previous work has demonstrated the use of Kolmogorov Complexity for Distributed Denial of Service (DDoS) attack detection [208].

$$(\hat{K}(x) \approx l(x)H)\left(\frac{x\#1}{x\#1 + x\#0}\right) + \log_2(l(x)) \tag{19}$$

$$H(p) = -p\log_2 p - (1.0 - p)\log_2 p - (1.0 - p) \tag{20}$$

$$H(X) \approx \sum_{l(x)=n} P(X = x)K(x) \tag{21}$$

Because Kolmogorov Complexity was originally derived for the study of randomness, it is interesting to note that randomness plays a significant role in the operation of the genetic algorithm itself. The initial genetic material should be generated randomly. Selection of genes for mutation and crossover points should also be done randomly. Finally, selection of gene pairs is done randomly, but in proportion to their fitness value.

Given the randomly generated nature of the initial genetic material, one would expect the complexity of the genetic material to decrease as the genetic algorithm evolves. This is clearly the case in the initial steep downward spike shown in Figure 130. As

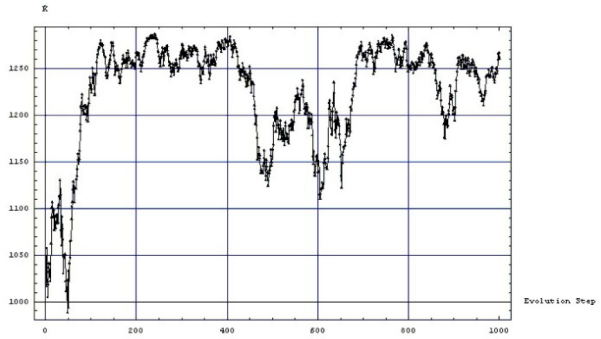

Figure 130. Complexity of Genetic versus Evolutionary Time Steps with Population 128.

the algorithm continues to evolve and the fitness of the genetic material improves, one would expect structure and order to appear. As mentioned earlier, in this specific case, the algorithm encourages the growth of binary strings that represent odd multiples of *0.5*.

Figure 131 shows the complexity, estimated as the

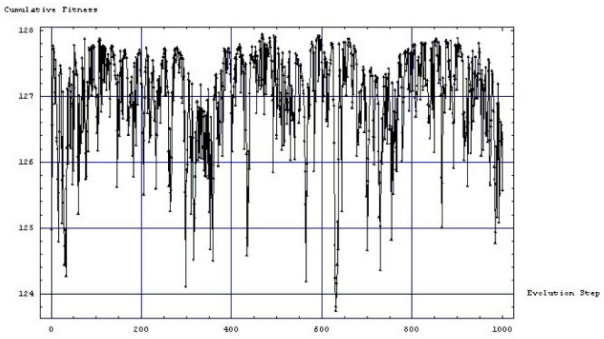

Figure 131. Cumulative Fitness Function of Genetic Material with Population 128.

compressed size, of the genetic material as a function of evolutionary steps. Compare with Figure 132, which shows the sum of the fitness values as a function of evolutionary steps. The complexity decreases as the cumulative fitness function increases, then rises again while evolution continues however, the fitness function does not significantly increase. The complexity measure seems to indicate that the first optimal genetic composition was found near evolution step 50. As the genetic algorithm continued beyond that point, the genetic material became more complex again with no corresponding benefit in fitness. This result was unanticipated, but is plausible as new solutions evolve, with varying complexity, attempting to maximize fitness.

The cumulative fitness function results (multiplied by 10 to shift upward for easier comparison with the estimated complexity) are shown in Figure 132. Note that the points of high complexity

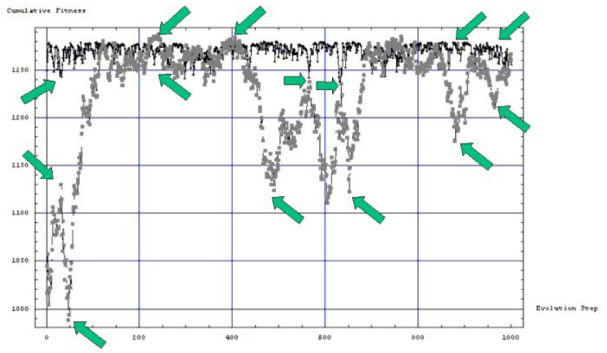

Figure 132. Complexity and Fitness Comparison.

always coincide with points of low cumulative fitness. Points of relatively low complexity correspond to high cumulative fitness. Arrows point to the extrema in the cumulative fitness function and estimated complexity that can be seen to align with extrema in the fitness function. In particular, minima in estimated complexity occur simultaneously with opposing maxima in the fitness. This indicates an inverse relationship between complexity and cumulative fitness extreme points.

Consider the complexity of the fitness function itself. The fitness function is an algorithmic representation of the fitness of a chromosome. The range resulting in maxima generated from the fitness function forms a string that represents the target complexity. In this particular genetic algorithm example, a solution of *0.5* for all *128* members of the population would yield an estimated complexity of *611.3*. This low a level of complexity was never reached for two reasons: there are multiple optimal solutions,

namely odd multiples of *0.5*, and the algorithm never exactly achieved odd multiples of *0.5*, but rather approximately close values. The remaining sections discuss how these concepts have been implemented to construct a fault tolerant network.

TOWARDS A SELF-EVOLVING NETWORK SYSTEM

Other papers from the Imperishable Networks Project have developed complexity-based techniques for fault detection and identification as discussed in [210] and [193]. The focus of this report is on progress towards self-composition of solutions assuming that other techniques, particularly complexity-based techniques, have identified faults. A problem with the genetic algorithm-based approach as previously described for use as a self-evolving system is that control is generally external to the genetic material and the genetic material is generally considered to be passive data. Instead the genetic material should be capable of being algorithmic information, that is, program code or objects. In addition, each chromosome, as an object, should contain the necessary capability to run the genetic algorithm. This would allow for a highly distributed and robust genetic algorithm capable of fault mitigation where the fault is represented through the fitness function.

A criticism of this approach might be that a genetically-engineered protocol stack will create a complex framework that will be difficult to understand and maintain. However, our approach is to compose the framework from simple components. Each of these components will be individually verifiable with respect to its properties and actions. As the components are arbitrarily composed to form a protocol stack, some protocol stacks may be generated that violate the principles of safety, consistency and correctness. One way to approach this is to define a fitness function that verifies the suitability of the stack with respect to the properties desired. Any mis-configured protocol stacks are automatically eliminated from consideration if the fitness function is carefully defined to check for the above-mentioned properties. However, this might make the definition of the fitness function itself cumbersome as every possible stack composition property will have to known a priori and an appropriate fitness “filter” defined. This will lead to a loss of elegance in the fitness function definition and consequently poor maintainability. A better approach would be to define syntactic and certain semantic composition properties in the individual components themselves, possibly in the form of logical expressions. These expressions will enforce constraints on the behavior of the components, which can be verified at run-time. The run-time system will embed a theorem-prover, which can be either a full-blown prover like PVS, NuPrl or SPIN or a reduced version of one, to systematically verify properties during composition itself. This reduces the burden on the programmer to define a proper fitness function that can catch and eliminate all types of composition errors.

Approach

Genetic material begins in a random state (*M*), and converges to the complexity of the optimal value produced by the fitness function. This enables true solution composition from a wide range of possible solutions. One problem with this approach is the time required evolving towards a feasible solution. Another problem is the fitness function itself has to be self-generated in some manner. Using Active function exists in the form of H_e where H_e is the estimatedVirtual Network Management Prediction [196], the fitness correct operation hypothesis of the system as described in [185].

In summary, the experiment in this section has shown a relationship among fitness, complexity, and the evolution of genetic material. Complexity estimation probes have been embedded in the General Electric Global Research Center Active Network testbed for use in security experimentation. The next section explains the framework developed to utilize the same complexity probes described in [208] to control the evolution of a genetic program within the active network. This makes the network highly resilient to faults by enabling the capability to adapt in a wide variety of ways.

GENETIC NETWORK PROGRAMMING ARCHITECTURE

The Magician Active Network [196] overlay network is used to test the feasibility of the genetically programmed network service concept. An active packet representing the nucleus (assuming network nodes are like eucaryotes- cells containing nuclei) is injected into all the `network nodes. The nucleus contains a population of chromosomes-- strings of functional units. Operation of Genetic Network Programming begins with the injection of basic building blocks, known as functional units, into the network

as shown in Figure 133. Cur rently, this “genetic

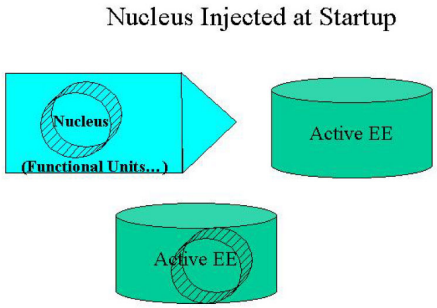

Figure 133. Injection of the Nucleus.

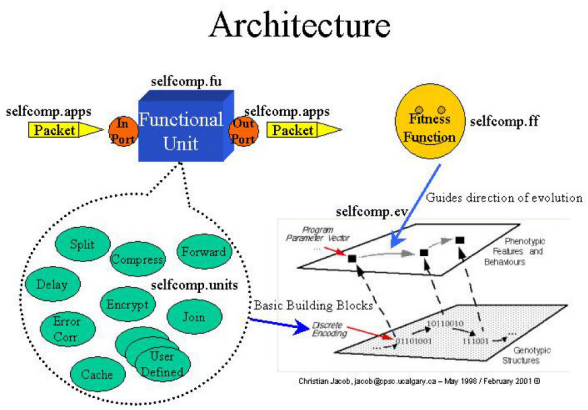

Figure 134. Functional Units, Evolution, and Fitness.

material” is flooded into each active node. However, the material will remain inactive in each active node until a fitness function is injected into the network. Receipt of a fitness function will cause evolution to proceed.

Functional units are very small pieces of code blocks that perform simple, well-defined operations upon an active packet. Examples of functional units are *Delay*, *Split*, *Join*, *Clone*, and *Forward*. There is also a *Null* functional unit whose use is explained later. Chromosomes are strings of functional units as shown in Figure 134. Once a chromosome is assembled, the codons can be translated into Amino Acids at the Ribosomes. In other words, the string of functional units will operate upon active packets from other applications (or other functional units) that traverse through the node. The chromosome is represented in the code in a form similar to a Lisp symbolic expression, for example: *((Null Join Split) (Delay Split Join Delay))*.

Mutation and recombination occur among a population of genes. Mutation is a probabilistic change of a functional unit to another functional unit. Recombination is the exchange of chromosome sections from two different chromosomes. In Figure 135, a close-up of a single node can be seen containing a very short chromosome strand.

A single incoming traffic stream, as shown in Figure 136 entering the center node, is split into multiple streams. Each stream is processed by a different chromosome. Note that currently in our implementation, the full traffic stream is split along

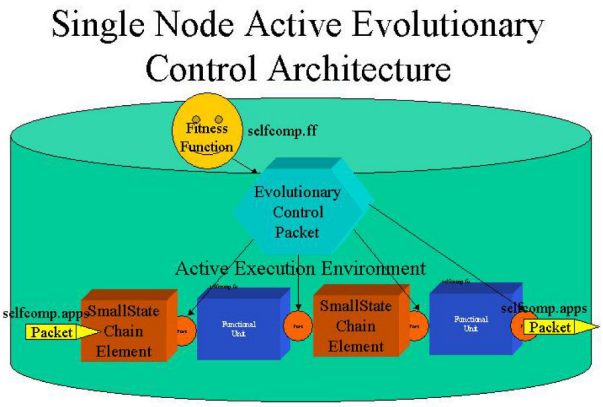

Figure 135. Single Node Genetic Programming Architecture.

each chromosome, however, it is hypothesized that traffic sampling could be used to reduce the overhead in creating the multiple streams.

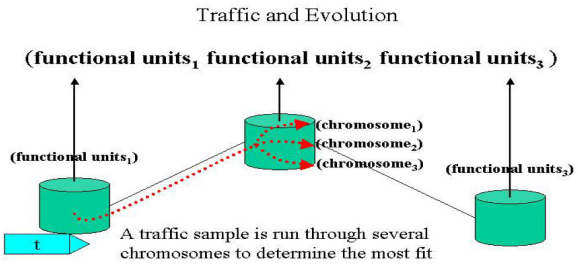

Figure 136. Breeding Traffic Streams.

As shown in Figure 137, fitness functions can be designed to measure quality at different layers of the traditional protocol stack. In this particular case, fitness measures are shown at the Transport, Network, and Link Layers. As a particular example, jitter control might have a fitness function that minimized per frame variance at the Link Layer. The Network Layer would attempt to maximize packet arrives at the destination in the reasonable time period, that is perform the routing function. The Transport Layer would have a fitness function that attempts to minimize end-to-end packet variance. The key is that each of these fitness functions need to work together towards reaching the stated goal in a reasonable manner. More will be said about the fitness function later.

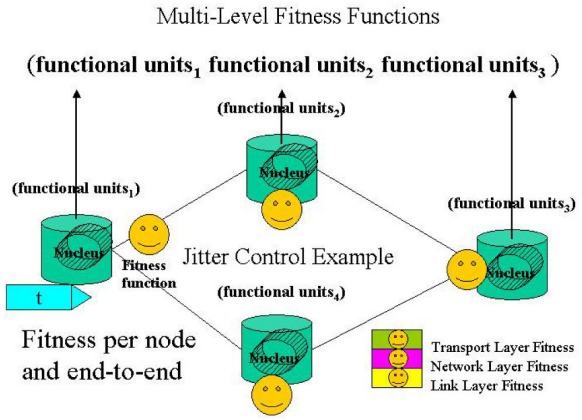

Figure 137. Multiple Levels of Fitness.

In Figure 138, recombination can occur both within a node or between two nodes. In addition, as shown in Figure 139, changing the route of a packet also effectively accomplishes a recombination because the packet processing will be dependent upon the genetic material at each node traversed.

A key component of the evolutionary process is the fitness function. Fitness functions are ``user'' defined and injected into the network to control the evolution of the genetic population. For example, in our initial tests, minimizing variance in transmission time was used as a simple fitness function. However, initial experiments quickly demonstrated that the design of the fitness function is the most critical element. It reminds one of the saying, ``Be careful of what you pray for..., because you might get it.'' Often the fitness was achieved, but in ways that were unexpected and sometimes detrimental to the intended

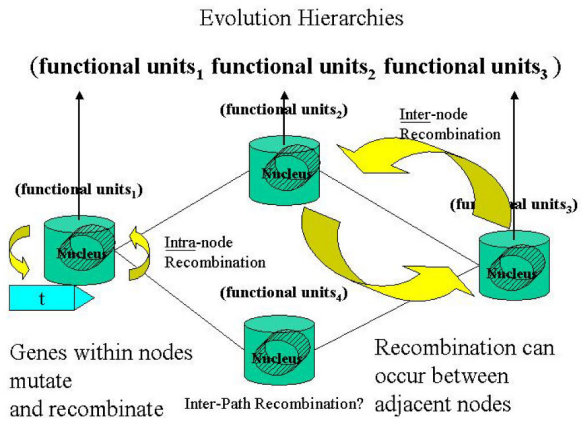

Figure 138. Recombination Levels.

operation of the network. As a trivial example, the variance can be minimized by slowing the traffic to a near halt. Thus, a low latency term had to be added to the jitter control fitness function.

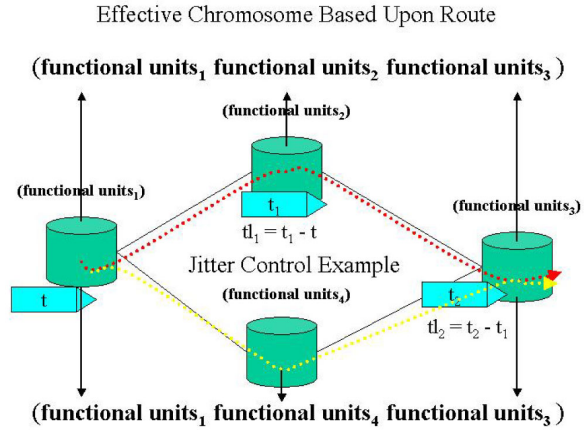

Figure 139. Chromosomes and Routing.

Genetically programmed active network jitter control

As a feasibility test, an adaptive jitter control mechanism was developed on a fixed, wired active communication network having the topology shown in Figure 139. The genetic algorithm was implemented as an active application in the Magician Active Network Execution Environment [196]. Packets originate from the left-most node in Figure 139and are destined for the right-most node in the figure. The dominant contributors to packet link transit time variability given the topology shown in Figure 139 are the fact that the active network is an overlay net-

work that has unspecified lower -layer traffic and that packets are loaded and executed within a Java Virtual Machine residing in each node and are subject to Java garbage collection which runs at unspecified times.

The fitness function on all nodes returns a greater fitness as the result of a Simple Network Management Protocol query of an Object Identifier that measures packet link transfer time variance on the destination node is minimized. As previously mentioned, the fitness function is itself an active packet that consists of an objective function. The function is highly general and can be comprised of any mathematical function of accessible metrics.

Figures 140 through Figure 143 show packet link transit variance through three of the chromosomes on the destination node and Figure 143 shows packet link transit variance without any jitter control mechanism at the destination node. Initial observation of the graphs shows that, overall, particularly as time progressed, the Chromosomes significantly reduced packet transit variance.

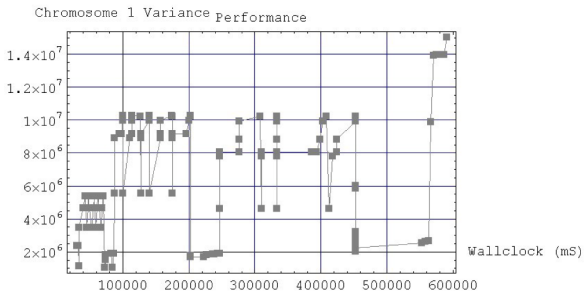

Figure 140. Packet Link Transit Variance (*milliseconds*^{2}) on Destination Node Through Chromosome One.

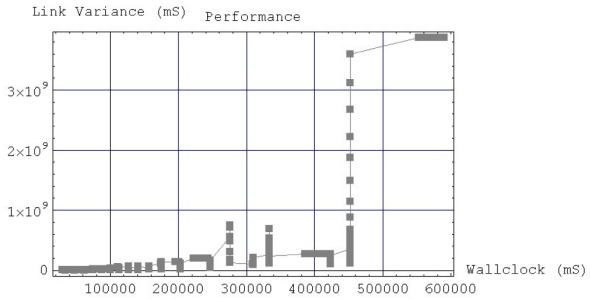

Figure 143. Packet Link Transit Variance (*milliseconds*^{2}) on Destination Node Without Jitter Control.

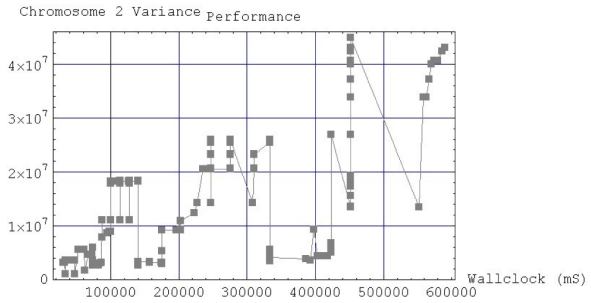

Figure 141. Packet Link Transit Variance (*milliseconds*^{2}) on Destination Node Through Chromosome Two.

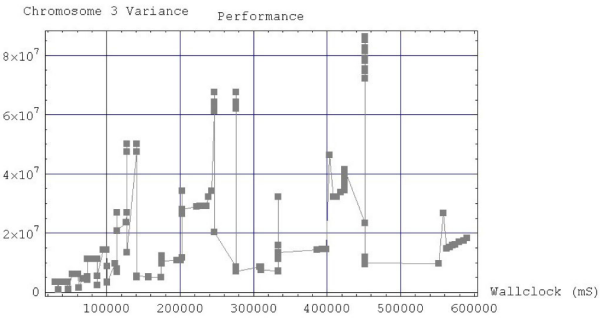

Figure 142. Packet Link Transit Variance (*milliseconds*^{2}) on Destination Node Through Chromosome Three.

Another observation of the experimental data is that the genetically programmed transit variance was initially worse than transit variance without any control mechanism. The reason for this is that the chromosomes begin operation with a random set of functional units and require time to converge to an optimal value.

Jitter control: a simple test case

While *a priori* techniques have been developed for jitter control in legacy networks, jitter control forms a simple, easily measured and controlled application for the network genetic programming technique. The functional units injected into the network should allow evolution of a variety of interesting solutions to reduce variance, including adding delays, forward along different paths, or perhaps new ideas that have not been thought of yet.

Conclusions

The most significant result from this project has been experimental validation that complexity plays a critical role in information assurance and can be broadly applied as the basis for security analysis and fault tolerant network design. Complexity Theory is a large and rapidly evolving science (Figure 1). As progress is made in various topics of Complexity Theory, the individual topics will help to re-enforce each other. For example, Vladimir Gudkov's results in the minimum dimensions required to characterize information flow could help develop a better Kolmogorov Complexity estimator. Our goal has been to reduce the requirement and dependence upon detailed a priori information about known attacks and detect novel attacks by computing vulnerability and detecting anomalous behavior based upon an inherent, fundamental property of information itself, namely, its complexity and sophistication. Results of complexity measures applied to network protocols, processes, and information have been presented and related to Information Assurance and network fault tolerance.

Accurate estimation of Kolmogorov Complexity is key to its usefulness in identifying correlation between attack flows. We have made progress in leveraging and developing in-line communication network complexity estimators and we are investigating and benchmarking more estimators for K(x). Estimates of Kolmogorov Complexity provide an objective parameter with which to provide information assurance through anomaly detection and objective model development. The capability of this metric is limited in part by the accuracy of its estimation, which must be traded against computational expense. The Optimal Symbol Compression Ratio complexity and sophistication algorithm may provide additional capability to discern anomalous behavior in information systems. Further research is needed to develop strategies for cost-effective use of this paradigm across entire systems.

With respect to the DDoS detection technique, its performance needs to be compared to more intelligent detection algorithms that are currently in use. In particular, its performance has to be measured in terms of resource tradeoffs, detection and false-alarm probability and response time. For example, the current technique performs its evaluation on the entire content of the packet. Anecdotal evidence has shown that performance degrades if the payload of the packet is encrypted and the size of the payload dominates the size of the packet. Techniques that adapt to payload size have been formulated and tested.

The next challenge is continuing the development of the K-Map (Kolmogorov Complexity Map of a system) and applying theory using Kolmogorov Complexity. For example, one significant application is identifying and controlling faults and DDoS attacks and tracing attacks back to the attacker. The fundamental hypothesis is that the attacker can be traced using a complexity-based approach because attacks must have a common pattern because they originate from a common source. We expect that advances in Complexity Theory, combined with reflective capability enabled by Active Networking, will enable significant advances in network fault tolerance and adaptation.

References

[1] *Combinatorial Foundations of Information Theory and the Calculus of Probabilities* Russian Math. Surveys, 38(4): 29-40, 1983.

[2] *Islands of Near-Perfect Self-Prediction* by Stephen F. Bush, Proceedings of VWsim'00: Virtual Worlds and Simulation Conference, WMC'00: 2000 SCS Western Multi-Conference, San Diego, SCS (2000) International Conference on Virtual Worlds and Simulation (VWSIM 2000), January 2000.

[3] Bush, Stephen F. and Kulkarni, Amit B., *Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework.* ISBN 0-306-46560-4, Kluwer Academic/Plenum Publishers. Spring 2001.

[4] Bush, Stephen F. and Kulkarni, Amit B. (GE), Galtier, Virginie and Carlinet, Yannick and Mills, Kevin L. (NIST), Ricciulli, Livio (Metanetworks*). Predicting and Controlling Resource Usage in an Active Network.* DARPA Active Networks PI Meeting, December 6-9, 2000, Atlanta, GA.

[5] Bush, Stephen, F. and Evans, *Kolmogorov Complexity for Information Assurance*, GE Corporate Research and Development Technical Report 2001CRD148.

[6] Evans, Scott, Bush, Stephen F., and Hershey, John, *Information Assurance through Kolmogorov Complexity*, DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) 12-14 June 2001, Anaheim, California.

[7] Evans, Scott and Bush, Stephen. F, *Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity*, Submitted to 2002 IEEE International Symposium on Information Theory.

[8] Kulkarni, Amit B. and Bush, Stephen F., *Active Network Management and Kolmogorov Complexity*, OpenArch 2001, Anchorage Alaska, April 27-28, 2001.

[9] Kulkarni, Amit B. and Bush, Stephen F., *Active Network. Management, Kolmogorov Complexity*, and Streptichrons by GE CRD Technical Report 2000CRD107, December 2000.

[10] Li, Ming and Vitányi, Paul. *An Introduction to Kolmogorov Complexity and Its Applications*, ISBN 0-387-94868-6, Springer, NY 1997.

[11] Liu, George Y. *The Effectiveness of a Full-Mobility Architecture for Wireless Mobile Computing and Personal Communications.* Ph.D. Thesis. Royal Institute of Technology, Stockholm, Sweden. March 1996.

[12] Virginie Galtier and Kevin L. Mills (NIST) and Yannick Carlinet and Stephen F. Bush and Amit Kulkarni. *Predicting Resource Demand in Heterogeneous Active Networks*, MILCOM 2001, McLean, VA, October 28-31.

[13] Virginie Galtier and Kevin L. Mills (NIST) and Yannick Carlinet and Stephen F. Bush and Amit Kulkarni. *Prediction and Controlling Resource Usage in a Heterogeneous Active Network*, Third Annual International Workshop on Active Middleware Services 2001, San Francisco, CA, August 6, 2001.

[14] Wallace, C. S. and Dowe, D. L., *Minimum Message Length and Kolmogorov Complexity*, The Computer Journal, Vol. 42, No 4. 1999.

[15] Bennett, C. *Thermodynamics of Computation.* International Journal of Physics, 21, 905-940.

[16] Bush, Stephen F. and Kulkarni, Amit B., *Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework.* ISBN 0-306-46560-4, Kluwer Academic/Plenum Publishers. Spring 2001.

[17] Bush, Stephen F. and Kulkarni, Amit B. (GE), Galtier, Virginie and Carlinet, Yannick and Mills, Kevin L. (NIST), Ricciulli, Livio (Metanetworks*). Predicting and Controlling Resource Usage in an Active Network.* DARPA Active Networks PI Meeting, December 6-9, 2000, Atlanta, GA.

[18] Bush, Stephen, F. and Evans, *Kolmogorov Complexity for Information Assurance*, GE Corporate Research and Development Technical Report 2001CRD148.

[19] Bush, Stephen, F. and Kulkarni, Amit B., and Evans, Scott C. *Active Virtual Network Management Prediction Enhancement via Kolmogorov Complexity Estimation*, Submitted to the 2002 IEEE Open Architecture Workshop. June 28-29, 2002, Hilton New York, New York City.

[20] Carothers, C. D. and Perumalla, K. S. and Fujimoto, R. M., *Efficient Optimistic Parallel Simulations using Reverse Computation*, pp. 126-135,

Proceedings of the 13th Workshop on Parallel and Distributed Simulation.

[21] Evans, Scott, Bush, Stephen F., and Hershey, John, *Information Assurance through Kolmogorov Complexity*, DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) 12-14 June 2001, Anaheim, California.

[22] Evans, Scott and Bush, Stephen. F, *Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity*, Submitted to 2002 IEEE International Symposium on Information Theory.

[23] Gacs, P., Tromp, J. T., and Vitányi, P. *Algorithmic Statistics*, IEEE Transactions on Information Theory, Vol 47, No 6, September 2001, pp. 2443-2463.

[24] Harmon, S. Y., DARPA Final Report, Volume I, *A Physical Model Of The Behavior Of Information Systems*, Zetetix, Oak Park, CA, October 2000.

[25] Hopcroft & Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[26] Hopcroft J.E., *An n Log n Algorithm for Minimizing the States in a Finite Automaton*, The Theory of Machines and Computations (Z. Kohavi, ed.), pp. 189-196, Academic Press, New York, 1971.

[27] Kirchher W., Li M., and Vitányi P., *The Miraculous Universal Distribution*, The Mathematical Intelligencer, Springer-Verlag, New York, Vol. 19, No. 4, 1997.

[28] Kulkarni, Amit B. and Bush, Stephen F., *Active Network Management and Kolmogorov Complexity*, OpenArch 2001, Anchorage Alaska, April 27-28, 2001.

[29] Kulkarni, Amit B. and Bush, Stephen F., *Active Network. Management, Kolmogorov Complexity*, and Streptichrons by GE CRD Technical Report 2000CRD107, December 2000.

[30] Li, Ming and Vitányi, Paul. *An Introduction to Kolmogorov Complexity and Its Applications*, ISBN 0-387-94868-6, Springer, NY 1997.

[31] Kulkarni, A., Minden, G., Hill, R., Wijata, Y., Sheth, S., Pindi, H., Wahhab, F., Gopinath, A. and Nagarajan, A. *Implementation of a Prototype Active Network*, IEEE OpenArch, San Francisco, 1998.

[32] Kulkarni, Amit B., Bush, Stephen, F., and Evans, Scott C., *Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics*. Submitted to the 2002 IEEE Open Architecture Workshop. June 28-29, 2002, Hilton New York, New York City.

[33] Liu, George Y. *The Effectiveness of a Full-Mobility Architecture for Wireless Mobile Computing and Personal Communications*. Ph.D. Thesis. Royal Institute of Technology, Stockholm, Sweden. March, 1996.

[34] Tennenhouse, D. L., Smith, J. M., Sincoskie W. D., Wetherall, D. J., and Minden, G. J. "A Survey of Active Network Research". *IEEE Communications Magazine*, 35(1): 80-86, Jan. 1997.

[35] Virginie Galtier and Kevin L. Mills (NIST) and Yannick Carlinet and Stephen F. Bush and Amit Kulkarni. *Predicting Resource Demand in Heterogeneous Active Networks*, MILCOM 2001, McLean, VA, October 28-31.

[36] Virginie Galtier and Kevin L. Mills (NIST) and Yannick Carlinet and Stephen F. Bush and Amit Kulkarni. *Prediction and Controlling Resource Usage in a Heterogeneous Active Network*, Third Annual International Workshop on Active Middleware Services 2001, San Francisco, CA, August 6, 2001.

[37] Wallace, C. S. and Dowe, D. L., *Minimum Message Length and Kolmogorov Complexity*, The Computer Journal, Vol. 42, No 4. 1999.

[38] Chapter 5: Emergent Protocols –Complexity As An Indicator of the On-Set Of Emergence for Developing Efficient Protocols For Low Capability Devices In Highly Dynamic Environments

[39] Bush, Stephen F. and Kulkarni, Amit B., "Proactive Network Management Using Active Networks," CRD Technical Report 2000CRD112, http://www.research.ge.com/crd_reports/index.htm

[40] Bush, Stephen F., "Active Virtual Network Management Prediction Project Final Report" funded by DARPA/ITO Contract Number: F30602-98-C- 0230 supported by the Air Force Research Laboratory/IF.

[41] Bush, Stephen F. and Kulkarni, Amit B. Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework. ISBN 0-306-46560-4. Kluwer Academic/Plenum Publishers. Spring 2001.

[42] Bush, Stephen F., Active Virtual Network Management Prediction, Parallel and Discrete Event Simulation Conference (PADS)'99, May 1999.

[43] Bush, Stephen F. and Barnett B., A Security Vulnerability Technique and Model, GE Corporate

Research and Development, T echnical Report 98CRD028, January 1998.
[44] Bush, Stephen F., Kulkarni A., Galtier V., Carlinet Y., Mills K. L. and Ricciulli L., Predicting and Controlling Resource Usage in an Active Network, DARPA Active Networks PI Meeting December 6-9, 2000, Atlanta, GA.
[45] Bush, Stephen F., Evans, Scott, Complexity-Based Information Assurance, To be submitted to Eighth ACM Conference on Computer and Communications Security (CCS-8), November 5-8, 2001, Philadelphia, Pennsylvania, USA.
[46] Coore D." Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer," Ph.D. Thesis, MIT Dept. of Electrical and Computer Science, Dec. 1998.
[47] J. P. Crutchfield. "The calculi of emergence: Computation, dynamics and induction," Physica D., vol. 75, no. 1–3. Pages 11–54, 1994.
[48] James P. Crutchfield and Melanie Mitchell. "The Evolution of Emergent Computation," Santa Fe Institute. 1994. Number 94-03-012.
[49] Evans, Scott, Bush, Stephen F., and Hershey, John, Information Assurance through Kolmogorov Complexity Accepted for publication at the DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) to be held 12-14 June 2001 in Anaheim, California.
[50] David A. Fisher and Howard F. Lipson. "Emergent Algorithms: A New Method for Enhancing Survivability in Unbounded Systems," Email: dfisher@cert.org and hfl@cert.org CERT® Coordination Center Software Engineering Institute.
[51] Edward Fredkin and Tommaso Toffoli. "Conservative Logic." International Journal of Theoretical Physics, vol. 21, pp. 219–53, 1982.
[52] Harmon, S. Y., A Physical Model Of The Behavior Of Information Systems, DARPA Final Report, Volume I, Zetetix, Oak Park, CA, October 2000.
[53] Kirchher W., Li M., and Vitányi P., The Miraculous Universal Distribution, The Mathematical Intelligencer, Springer-Verlag, New York, Vol. 19, No. 4, 1997.
[54] Amit Kulkarni & Gary Minden. "Active Network Services for Wired/Wireless Networks," INFOCOM'99, NY, March 1999
[55] Kulkarni, A. B. and Bush, Stephen F., Active Network Management, Kolmogorov, Complexity, and Streptichrons. GE-CRD Class I Technical Report 2000CRD107 (www.research.ge.com/crd_reports/index.htm).
[56] Kulkarni A. B., Minden G. J., Hill R., Wijata Y.,Sheth S., Pindi H., Wahhab F., Gopinath A., and Nagarajan A, Implementation of a Prototype Active Network, OPENARCH'98, 1998.
[57] M. Mitchell, J. P. Crutchfield and P. T. Hraber. "Evolving Cellular Automata to Perform Computations," Physica D. 1993.
[58] Ming Li and Paul Vitányi, Introduction to Kolmogorov Complexity and its Applications, Spreinger-Verlag, 1993. ISBN 0-387-94053-8.
[59] Rose, M. T., The Simple Book, An Introduction to the Management of TCP/IP Based Internets, Prentice Hall, 1991.
[60] Smart Dust Performer: University of California, Berkeley, Dr. Kristofer Pister (510) 643-9268 Agent: AIC Mr. Roy Peters (520) 538–409.
[61] Tennenhouse, D. L., Smith, J. M., Sincoskie W. D., Wetherall, D. J., and Minden, G. J., A Survey Of Active Network Research, IEEE Communications Magazine, 35(1): 80–86, Jan. 1997.
[62] Tinker P. and Agra J., Adaptive Model Prediction Using Time Warp, SCS'90, 1990.
[63] Wallace, C. S. and Dowe, D. L., Minimum Message Length and Kolmogorov Complexity, The Computer Journal, Vol. 42, No 4, 1999.
[64] Wojciech Zurek. Complexity, Entropy and the Physics of Information. Addison-Wesley, 1990. ISBN 0-201-51509-1.
[65] http://www.swarm.org
[66] Bush, Stephen F. and Kulkarni, Amit B., *Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework.* ISBN 0-306-46560-4, Kluwer Academic/Plenum Publishers. Spring 2001.
[67] Bush, Stephen F. and Kulkarni, Amit B. (GE), Galtier, Virginie and Carlinet, Yannick and Mills, Kevin L. (NIST), Ricciulli, Livio (Metanetworks*). Predicting and Controlling Resource Usage in an Active Network.* DARPA Active Networks PI Meeting, December 6-9, 2000, Atlanta, GA.
[68] Bush, Stephen, F. and Evans, *Kolmogorov Complexity for Information Assurance*, GE Corporate Research and Development Technical Report 2001CRD148.
[69] Bush, Stephen, F. and Kulkarni, Amit B., and Evans, Scott C. *Active Virtual Network Manage-*

ment Prediction Enhancement via Kolmogorov Complexity Estimation, Submitted to the 2002 IEEE Open Architecture Workshop. June 28-29, 2002, Hilton New York, New York City.

[70] Bush, Stephen, F. and Kulkarni, Amit B., and Evans, Scott C. *Self-Generating Anti-Faults Enabled by Kolmogorov Complexity for Self-Healing Systems*, Not yet published.

[71] Carothers, C. D. and Perumalla, K. S. and Fujimoto, R. M., *Efficient Optimistic Parallel Simulations using Reverse Computation*, pp. 126-135, Proceedings of the 13th Workshop on Parallel and Distributed Simulation.

[72] Evans, Scott, Bush, Stephen F., and Hershey, John, *Information assurance through Kolmogorov Complexity*, DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) 12-14 June 2001, Anaheim, California.

[73] Evans, Scott and Bush, Stephen. F, *Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity*, Submitted to 2002 IEEE International Symposium on Information Theory.

[74] Goldberg, David E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989, Addison-Wesley.

[75] Goldberg, David E., *Genetic and Evolutionary Algorithms Come of Age*. Communications of the ACM, March 1994, pp 113-119.

[76] I. De Falco, M. Conte, A. Della Cioppa, E. Tarantino and G. Trautteur, *Genetic Programming Estimates of Kolmogorov Complexity*, pp. 743-750, ISBN 1-55860-487-1, Proceedings of the 7th International Conference on Genetic Algorithms, July 19-23, San Francisco, 1997.

[77] Gacs, P., Tromp, J. T., and Vitányi, P. *Algorithmic Statistics*, IEEE Transactions on Information Theory, Vol 47, No 6, September 2001, pp. 2443-2463.

[78] Harmon, S. Y., DARPA Final Report, Volume I, *A Physical Model Of The Behavior Of Information Systems*, Zetetix, Oak Park, CA, October 2000.

[79] Hopcroft & Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[80] Hopcroft J.E., *An n Log n Algorithm for Minimizing the States in a Finite Automaton*, The Theory of Machines and Computations (Z. Kohavi, ed.), pp. 189-196, Academic Press, New York, 1971.

[81] Kirchher W., Li M., and Vitányi P., *The Miraculous Universal Distribution*, The Mathematical Intelligencer, Springer-Verlag, New York, Vol. 19, No. 4, 1997.

[82] Kulkarni, Amit B. and Bush, Stephen F., *Active Network Management and Kolmogorov Complexity*, OpenArch 2001, Anchorage Alaska, April 27-28, 2001.

[83] Kulkarni, Amit B. and Bush, Stephen F., *Active Network. Management, Kolmogorov Complexity*, and Streptichrons by GE CRD Technical Report 2000CRD107, December 2000.

[84] Li, Ming and Vitányi, Paul. *An Introduction to Kolmogorov Complexity and Its Applications*, ISBN 0-387-94868-6, Springer, NY 1997.

[85] Liu, George Y. *The Effectiveness of a Full-Mobility Architecture for Wireless Mobile Computing and Personal Communications*. Ph.D. Thesis. Royal Institute of Technology, Stockholm, Sweden. March, 1996.

[86] Srinivas, M., Patnaik, Latit M., *Genetic Algorithms: A Survey*. IEEE Computer, June 1994, pp 17-26.

[87] Tennenhouse, D. L., Smith, J. M., Sincoskie W. D., Wetherall, D. J., and Minden, G. J. "A Survey of Active Network Research". *IEEE Communications Magazine*, 35(1): 80-86, Jan. 1997.

[88] Virginie Galtier and Kevin L. Mills (NIST) and Yannick Carlinet and Stephen F. Bush and Amit Kulkarni. *Predicting Resource Demand in Heterogeneous Active Networks*, MILCOM 2001, McLean, VA, October 28-31.

[89] Virginie Galtier and Kevin L. Mills (NIST) and Yannick Carlinet and Stephen F. Bush and Amit Kulkarni. *Prediction and Controlling Resource Usage in a Heterogeneous Active Network*, Third Annual International Workshop on Active Middleware Services 2001, San Francisco, CA, August 6, 2001.

[90] Wallace, C. S. and Dowe, D. L., *Minimum Message Length and Kolmogorov Complexity*, The Computer Journal, Vol. 42, No 4. 1999.

[91] I. Kuntz, J. Blaney, S. Oatley, R. Langridge, and T. Ferrin, *A geometric approach to macromolecule-ligand interaction*s, Journal of Molecular Biology, 161: 269-288, 1982.

[92] B. Shoichet, D. Bodian, and I. Kuntz, *Molecular docking using shape descriptor*s, Journal of Comp. Chemistry, 13(3): 380-397, 1992.

[93] E. Meng, D. Gschwend, J. Blaney, and I. Kuntz, *Orientational sampling and rigid-body minimization*

in molecular dockin g, Proteins, 17(3): 266-278, 1993.
[94] Kirchher W., Li M., and Vitányi P., The Miraculous Universal Distribution, The Mathematical Intelligencer, Springer-Verlag, New York, Vol. 19, No. 4, 1997.
[95] Albert-Laszlo Barabasi, Vincent W. Freeh, Hawoong Jeong & Jay B. Brockman, Parasitic Computing, Nature, Vol. 412, 30 August 2001, www.nature.com, pages 894-897.
[96] Aris Zakinthinos. On The Composition of Security Properties. University of Toronto, 1997. Ph.D. Thesis.
[97] Li, Ming and Vitányi, Paul An Introduction to Kolmogorov Complexity and Its Applications, Springer, NY 1997.
[98] Shannon, C.E., "A mathematical theory of communication," Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October 1948
[99] C. Bennett, P. Gacs, M. Li, P. Vitányi, and W. Zurek, Information Distance, IEEE Transactions on Information Theory, Vol. 44, July 1998.
[100] Walter Rudin. Principles of Mathematical Analysis. McGraw-Hill, 1953. ISBN 0-07-054235-X.
[101] Stephen Wolfram, Mathematica...A System for Doing Mathematics by Computer, Addison-Wesley, Reading, MA, USA, second edition, 1991.
[102] Bush, Stephen F. and Kulkarni, Amit B. Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework. ISBN 0-306-46560-4. Kluwer Academic/Plenum Publishers. Spring 2001.
[103] Evans, Scott and Bush, Stephen. F, Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity, GE Research and Development Technical Report 2001CRD159, www.research.ge.com/~bushsf/ftn.
[104] Kulkarni, Amit B., Bush, Stephen, F., and Evans, Scott C., Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics. Submitted to the 2002 IEEE Computer Security Foundations Workshop. June 24-26, 2002 Keltic Lodge, Cape Breton, Nova Scotia, Canada
[105] Bush, Stephen F. and Barnett, Bruce. A Security Vulnerability Technique and Model. GE Corporate Research and Development, January 1998. Technical Report 98CRD028.
[106] David G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley, 1989.
[107] Theory, Vol. 44, July 1998. John D. Howard, An Analysis of Security Incidents on the Internet 1989-1995, Ph.D. thesis, Carnegie Mellon University, Apr. 1997.
[108] Bush, Stephen F. and John Hershey and Kirby Vosburgh. Brittle Systems Analysis. http://xxx.lanl.gov/, 1999.
[109] Bush, S. F., "Active Virtual Network Management Prediction: Complexity as a Framework for Prediction, Optimization, and Assurance", IEEE Computer Society Press in the proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), to be held May 29-31, 2002, in San Francisco, California, USA.
[110] A. M. Turing. "On Computable Numbers with an Application to the Entscheidungsproblem." Proceedings of the London Math Society, Ser. 2, 42:230-265, 1936.
[111] Adam Shwartz and Alan Weiss. Large Deviations for Performance Analysis. Chapman and Hall, 1995. ISBN 0-412-06311-5.
[112] Alfred Aho, John Hopcroft, and Jeffery Ullman. The Design and Analysis of Computer Algorithms. ISBN 0-201-00029-6.
[113] Amir Dembo and Ofer Zeitouni. Large Deviations Techniques and Applications. Springer, 1998. ISBN 0-387-98406-2.
[114] Aris Zakinthinos. On The Composition of Security Properties. University of Toronto, 1997. Ph.D. Thesis.
[115] Bush, Stephen F. and Barnett, Bruce. A Security Vulnerability Technique and Model. GE Corporate Research and Development, January 1998. Technical Report 98CRD028.
[116] Bush, Stephen F. and Kulkarni, Amit B. Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework. ISBN 0-306-46560-4. Kluwer Academic/Plenum Publishers. Spring 2001.
[117] C. Bennett, P. Gacs, M. Li, P. Vitányi, and W. Zurek, *Information Distance*, IEEE Transactions on Information Theory, Vol. 44, July 1998.
[118] Cover, T. M. and Thomas, J. A. Elements of Information Theory. Wiley, NY, 1991.
[119] D. Denning, P. Denning, Internet Besieged, Addison Wesley, Mass, 1998.

[120]D. Eastlake, J. Schiller, S. Crocker "Randomness Requirements for Security", Internet draft, 30-Nov-00.
[121]David G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley, 1989.
[122]Denning, Elizabeth R, Cryptography and Data Security, Addison-Wesley, Mass, 1982.
[123]Evans, Scott, Bush, Stephen F., and Hershey, John, Information assurance through Kolmogorov Complexity Accepted for publication at the DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001) to be held 12-14 June 2001 in Anaheim, California.
[124]Fraundorf, P. "Heat Capacity in Bits," April 28, 2000, Downloaded from URL http://www.umsl.edu/~fraundor/ifzx/cvinbits.html. An active revision of cond-mat/9711074 in the Los Alamos archives.
[125]Frieden, Roy, B. Physics from Fisher Information. ISBN 0-521-63167-X. Cambridge University Press. 1999.
[126]G. J. Chaitin. The Limits of Mathematics. Lecture Notes in Computer Science, volume 888, 1995, ISSN 0302-9743.
[127]Giancoli, Douglas C. *General Physics,* Prentice Hall, INC, Englewood Cliffs, NJ.
[128]Harmon, S. "A Physical Model of the Behavior of Information Systems," DARPA Project: Exploring a Theory Describing the Physics of Information Systems, Final Report, Volume I. October 2000.
[129]Hopcroft & Ullman, "Introduction to Automata Theory, Languages, and Computation". Addison Wesley, 1979.
[130]Hopcroft J.E., "An n Log n algorithm for minimizing the states in a finite automaton", The Theory of Machines and Computations (Z. Kohavi, ed.), pp. 189-196, Academic Press, New York, 1971.
[131]C. Bennett, P. Gacs, M. Li, P. Vitányi, and W. Zurek, Information Distance, IEEE Transactions on Information
[132]Theory, Vol. 44, July 1998. John D. Howard, An Analysis of Security Incidents on the Internet 1989-1995, Ph.D. thesis, Carnegie Mellon University, Apr. 1997.
[133]Kirchher W., Li M., and Vitányi P., The Miraculous Universal Distribution, The Mathematical Intelligencer, Springer-Verlag, New York, Vol. 19, No. 4, 1997.
[134]Li, Ming and Vitányi, Paul An Introduction to Kolmogorov Complexity and Its Applications, Springer, NY 1997.
[135]Masud Mansuripur. Introduction to Information Theory. Prentice Hall, 1987. ISBN 0-13-484668-0.
[136]Ming Li and Paul Vitányi. Introduction to Kolmogorov Complexity and Its Applications. Springer-Verlag, 1993. ISBN 0-387-94053-7.
[137]Rolf Herken, Ed. The Universal Turing Machine, A Half-Century Survey. Springer-Verlag, NY, 1995.
[138]Stephen F. Bush and John Hershey and Kirby Vosburgh. Brittle Systems Analysis. http://xxx.lanl.gov/, 1999.
[139]Stephen Wolfram, Mathematica...A System for Doing Mathematics by Computer, Addison-Wesley, Reading, MA, USA, second edition, 1991.
[140]Swarm model downloaded from URL http://www.swarm.org.
[141]Tennenhouse, D. L., Smith, J. M., Sincoskie W. D., Wetherall, D. J., and Minden, G. J. "A survey of active network research". IEEE Communications Magazine, 35(1): 80-86, Jan. 1997.
[142]W. Kirchher, M. Li, and P. Vitányi. The Miraculous Universal Distribution. The Mathematical Intelligencer, Springer-Verlag, New York, Vol. 19, No. 4, 1997.
[143]Wallace, C. S. and Dowe, D. L., *Minimum Message Length and Kolmogorov Complexity*, The Computer Journal, Vol. 42, No 4, 1999.
[144]Walter Rudin. Principles of Mathematical Analysis. McGraw-Hill, 1953. ISBN 0-07-054235-X.
[145]Wojciech Zurek. Complexity, Entropy and the Physics of Information. Addison-Wesley, 1990. ISBN 0-201-51509-1.
[146]Zamir Bavel. Introduction to the Theory of Automata. ZB Publishing Industries, 1993. ISBN 0-9623885-0-5.
[147]Rivest, R. L., Shamir, A. and Adleman, L. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems" Communications of the ACM, February 1978, Volume 21, Number 2.
[148]Diffie, W., and Hellman, M. "New directions in cryptography." IEEE Transactions on Information Theory IT-22, 6 (Nov. 1976), 644-654.

[149]Gacs, P ., Tromp, J. T., and Vitányi, P. *Algorithmic Statistics*, IEEE Transactions on Information Theory, Vol. XX, No Y, Month 2001.

[150]Diffie, W. and Hellman, M "Privacy and Authentication: An Introduction to Cryptography" Proceedings of the IEEE, Vol. 67, No 3. March 1979.

[151]C. E. Shannon, "A mathematical theory of communication," Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October 1948.

[152]Albert-Laszlo Barabasi, Vincent W. Freeh, Hawoong Jeong & Jay B. Brockman, *Parasitic Computing*, Nature, Vol. 412, 30 August 2001, www.nature.com, pages 894-897.

[153]Evans, Scott and Bush, Stephen. F, Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity, Submitted to 2002 IEEE International Symposium on Information Theory.

[154]Kulkarni, Amit B., Bush, Stephen, F., and Evans, Scott C., Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics. Submitted to the 2002 IEEE Open Architecture Workshop. June 28-29, 2002, Hilton New York, New York City.

[155]Bush, Stephen, F. and Kulkarni, Amit B., and Evans, Scott C. *Active Virtual Network Management Prediction Enhancement via Kolmogorov Complexity Estimation*, Submitted to the 2002 IEEE Open Architecture Workshop. June 28-29, 2002, Hilton New York, New York City.

[156]Gil, T. and Poletto, M. "MULTOPS: a data structure for bandwidth attack detection, "USENIX 2001.

[157]Bazek, R., Kim, H., Rozovskii, B., and Tartakovsky, A. "A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point methods," IEEE Systems, Man and Cybernetics Information Assurance Workshop, June 2001.

[158]Li, M. and Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, 1997.

[159]Kulkarni, A., Minden, G., Hill, R., Wijata, Y., Sheth, S., Pindi, H., Wahhab, F., Gopinath, A. and Nagarajan, A. "Implementation of a Prototype Active Network," IEEE OpenArch, San Francisco, 1998.

[160]Bush, Stephen F. and Kulkarni, Amit B., *Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework*. ISBN 0-306-46560-4, Kluwer Academic / Plenum Publishers. Spring 2001.

[161]Evans, S, Bush, S. F., and Hershey, J., "Information Assurance through Kolmogorov Complexity", DARPA Information Survivability Conference & Exposition II, 2001, Proceedings Vol 2, pp 322-331.

[162]Evans, S. and Bush, S. F. "Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity", submitted to 2002 IEEE International Symposium on Information Theory, June 30 – July 5, 2002.

[163]V. Gudkov, J. E. Johnson and S. Nussinov, "Approaches to Network Classification", arXiv: cond-mat / 0209111 (2002).

[164]V. Gudkov and J. E. Johnson, "Multidimensional Network Monitoring for Intrusion Detection", arXiv: cs.CR / 0206020, 2002.

[165]V. Gudkov and J. E. Johnson, "New approach for network monitoring and intrusion detection", arXiv: cs.CR / 0110019, 2001.

[166]V. Gudkov and J. E. Johnson, "Network as a Complex System: Information Flow Analysis, arXiv: nlin.CD / 0110008, 2001.

[167]Barron, A., Rissanen, J. and Yu, B. "The Minimum Description Length Principle in Coding and Modeling," IEEE Transactions on Information Theory, Vol 44, No 6. October 1998.

[168]Cover, T. M. and Thomas, J. A. Elements of Information Theory. Wiley, NY, 1991.

[169]Evans, S, Bush, S. F., and Hershey, J., "Information Assurance through Kolmogorov Complexity", DARPA Information Survivability Conference & Exposition II, 2001, Proceedings Vol 2, pp 322-331.

[170]Evans, S. C, Barnett, B., and Bush, S. F. `Complexity Mapping for Information Assurance and Detection of FTP Exploits," Unpublished.

[171]Evans, S. C. Barnett, B. "Conservation of Complexity for Network Security", accepted for publication in MILCOM 2002.

[172] Gacs, P. Tromp, J. and Vitányi, P. "Algorithmic Statistics" IEEE Transactions on Information Theory, 47:6(2001), 2443-2463.

[173]Kieffer, J. C. and Yang, E. "Sequential Codes, Lossless Compression of Individual Sequences, and Kolmogorov Complexity," IEEE Transactions of Information Theory, Vol 42, 1 January 1996.

[174] Kosaraju, S. R. and Manzini, G. "Compression of Low Entropy Strings with Lempel-Ziv Algorithms" SIAM J. COMPUT. Vol 29, No 3. pp 893-911.

[175]Kulkarni, A. B., Bush, S. F. and Evans, S. C. ``Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics," GE Research Technical Report 2001CRD176. December 2001.

[176]Lempel, A. and Ziv, J. "On the Complexity of Finite Sequences," IEEE Transactions of Information Theory, Vol IT 22, January 1976, pp 75-81.

[177]Li, M. and Vitányi, P. An Introduction to Kolmogorov Complexity and Its Applications, Springer, NY 1997.

[178]Powell, D. R, Dowe, D. L., Allison, L. and Dix, T. I. "Discovering simple DNA sequences by compression", Monash University Technical Report, monash.edu.au.

[179]Shannon, C. E. ``A mathematical Theory of Communication," Bell Systems Technical Journal, Vol 27, pp. 379-423, 623-656, October, 1948.

[180]Bush, Stephen F., "Active Virtual Network Management Prediction: Complexity as a Framework for Prediction, Optimization, and Assurance" Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), IEEE Computer Society Press, pp. 534-553, ISBN 0-7695-1564-9, May 29-30, 2002, San Francisco, California, USA.

[181]Vitányi, P. "Meaningful Information," unpublished. 2002 http://www.cwi.nl/ paulv.

[182] Wallace, C. S. and Dowe, D. L., "Minimum Message Length and Kolmogorov Complexity," The Computer Journal, Vol. 42, No 4. 1999.

[183]Ziv, J. and Lempel, A. "Compression of individual sequences via variable length coding," IEEE Trans. Inform. Theory, vol IT-24, pp. 530-536, 1978.

[184]http://www.gzip.org/zlib/.

[185]Stephen F. Bush, "Active Virtual Network Management Prediction: Complexity as a Framework for Prediction, Optimization, and Assurance," in IEEE Computer Society Press, Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA, May 2002, pp. 534–553, ISBN 0-7695-1564-9.

[186]S. Bhattacharjee, K. Calvert, Y Chae, S. Merugu, M. Sanders, and E. Zegura, "CANEs: An Execution Environment for Composable Services," in IEEE Computer Society Press, Proceedings of the 2002 DARPA ActiveNetworks Conference and Exposition (DANCE 2002), San Francisco, CA, May 2002, pp. 255–272.

[187]G. Minden, E. Komp, M. Kannan, S. Subramaniam, S. Tan, S. Vallabhaneni, and J. Evans, "Composite Protocols for Innovative Active Services," in IEEE Computer Society Press, Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA, May 2002, pp. 157–164.

[188]I. Kuntz, J. Blaney, S. Oatley, R. Langridge, and T. Ferrin, "A geometric approach to macromolecule-ligand interactions," *Journal of Molecular Biology*, 1982.

[189]B. Shoichet, D. Bodian, and I. Kuntz, "Molecular docking using shape descriptors," *Journal of Comp. Chemistry*, 1992.

[190]E. Meng, D. Gschwend, J. Blaney, and I. Kuntz, "Orientational sampling and rigid-body minimization in molecular docking," *Proteins*, pp. 266–278, 1993.

[191]Ming Li and Paul Vitányi, *Introduction to Kolmogorov Complexity and its Applications.*, Springer-Verlag, Aug. 1993.

[192]Amit B. Kulkarni and Stephen F. Bush, "Active network management, kolmogorov complexity, and streptichrons," Tech. Rep. 2000CRD107, General Electric Corporate Research and Development, Dec. 2000, http://www.research.ge.com/~bushsf/ftn.

[193]Amit B. Kulkarni and Stephen F. Bush, "Active network management and kolmogorov complexity," in *Proceedings of IEEE OpenArch 2001*, Apr. 2001.

[194]Stephen F. Bush and Scott C. Evans, "Kolmogorov complexity for information assurance," Tech. Rep. 2001CRD148, General Electric Corporate Research and Development, 2001, http://www.research.ge.com/~bushsf/ftn.

[195]Scott C. Evans, Stephen F. Bush, and John E. Hershey, "Information assurance through kolmogorov complexity," in *DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001)*, June 2001, vol. II, pp. 322–331, http://www.research.ge.com/~bushsf/ftn.

[196]Stephen F. Bush and Amit B. Kulkarni, *Active Networks and Active Network Management: A Proactive Management Framework*, Kluwer Academic/Plenum Publishers, ISBN 0-306-46560-4,

2001,
http://www.research.ge.com/~bushsf/ftn.
[197]James M. Bower and Hamid Bolouri, *Computational Modeling of Genetic and Biochemical Networks*, The MIT Press, 2001, 0-262-02481-0.
[198]M. Conte, G. Tautteur, I. De Falco, A. Della Cioppa, and E. Tarantino, "Genetic programming estimates of kolmogorov complexity," in *Genetic Algorithms: Proceedings of the Seventh International Conference*, Thomas Back, Ed., Michigan State University, East Lansing, MI, USA, 19-23 1997, pp. 743–750, Morgan Kaufmann.
[199]I. De Falco, A. Iazzetta, E. Tarantino, A. Della Cioppa, and G. Trautteur, "A kolmogorov complexity-based genetic programming tool for string compression," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, Eds., Las Vegas, Nevada, USA, 10-12 2000, pp. 427–434, Morgan Kaufmann.
[200]Byoung-Tak Zhang and Heinz Mühlenbein, "Balancing accuracy and parsimony in genetic programming," *Evolutionary Computation*, vol. 3, no. 1, pp. 17–38, 1995.
[201]Peter Nordin and Wolfgang Banzhaf, "Programmatic compression of images and sound," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, Eds., Stanford University, CA, USA, 28–31 1996, pp. 345–350, MIT Press.
[202]C. S. Wallace and D. L. Dowe, "Minimum message length and Kolmogorov complexity," *The Computer Journal*, vol. 42, no. 4, pp. 270–283, 1999.
[203]C. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low memory, modular time warp system," 2000.
[204]David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
[205]David E. Goldberg, "Genetic and evolutionary algorithms come of age," *Communications of the ACM*, pp. 113ñ119, 1994.
[206]M. Srinivas and Latit M. Patnaik, "Genetic algorithms: A survey, "IEEE *Computer*, pp. 17ñ26, 1994.
[207]Scott Evans and Bruce Barnett, "Network security through conservation of complexity," in *MILCOM*, The Disneyland Resort, Anaheim, CA, USA, Oct 2002, IEEE.
[208]Amit B. Kulkarni, Stephen F. Bush, and Scott C. Evans, "Detecting distributed denial-of-service attacks using kolmogorov complexity metrics," Tech. Rep. 2001CRD176, GE Global Research Center, Dec. 2001.
[209]Scott C. Evans and Stephen F. Bush, "Symbol compression ratio for string compression and estimation of kolmogorov complexity," Tech. Rep. 2001CRD159, General Electric Corporate Research and Development, 2001, http://www.research.ge.com/~bushsf/ftn.
[210]Stephen F. Bush and Scott C. Evans, "Complexity-based information assurance," Tech. Rep. 2001CRD084, General Electric Corporate Research and Development, Oct. 2001, http://www.research.ge.com/~bushsf/ftn.
[211]Sipser, M. Introduction to the Theory of Computation, PWS Publishing C, Boston, 1996.
[212]Crutchfield, J. "Reconstructing Language Hierarchies," *Information Dynamics*, Atmanspracher, H. A. et al. ed. Plenum Press, New York 1990.
[213]Mitchell, T. *Machine Learning*, McGraw-Hill, 1997.
[214]http://www.its.bldrdoc.gov/projects/t1glossary2000/
[215]Barron, A., Rissanen, J. and Yu, B. "The Minimum Description Length Principle in Coding and Modeling," *IEEE Transactions on Information Theory*, Vol 44, No 6. October 1998.

Appendix A
Operation of the Network Insecurity Path Analysis Tool (NIPAT)

Network Insecurity Path Analysis Tool (NIPAT) allows various types of node groupings in order to help visualize the vulnerability paths.

In Figure 144, all object types are grouped together. The nodes could also be grouped by such characteristics as hostname or sub-network. In Figure 145, the vulnerabilities that have been identified and grouped as vectors to vulnerability targets have been expanded to show more detail about the individual vulnerabilities.

In Figure 146, 40 parent objects of **sun4nbin** are grouped within a single node. Also note that the **root** account is clearly visible as reachable through the vulnerability path. The next paragraph provides an example of analyzing a vulnerability graph that provides a quick introduction to more of NIPAT's capabilities.

One of the security assessment operations NIPAT can perform is to determine the vulnerability of a particular entity given an attack on a particular node. The target entity **Host C Vulnerability 4** is identified by a white cross hair in Figure 147, and the attacking node is labeled **Attacker** with a flow identified by the label of its connecting path. The optimal vulnerability path is the sum of flows into node Host C Vulnerability 4, as shown in Figure 148. It has flow strength of 6.0. In Figure 149 the optimal path that the attacker can take to reach the target is shown in yellow. Thus NIPAT provides the ability to examine how the placement of security safeguards such as intrusion detectors within the network affect total network security. In effect, this tool becomes a security-modeling tool, where one can experiment with the placement of security safeguards representing such entities as firewalls, intrusion detectors, and access lists. These can be positioned at various locations in order to determine network security.

There are two main algorithms that can be run in NIPAT; the first is a probabilistic analysis and the second is a maximum flow analysis. Let us start with the probabilistic analysis. *Select a node to be the target of the attack by clicking on the* ***Select Nodes*** *toggle button*. Then select a node; in this case we have selected **Host C Vuln 4**. A white cross hair will appear over the node to indicate it has been selected. *Choose* ***Algorithms*** *and*

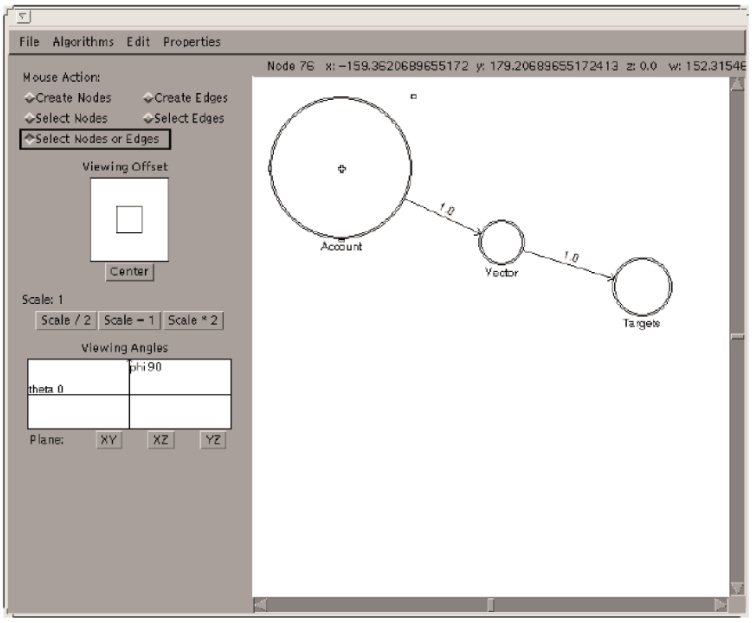

Figure 144. Attack Vectors.

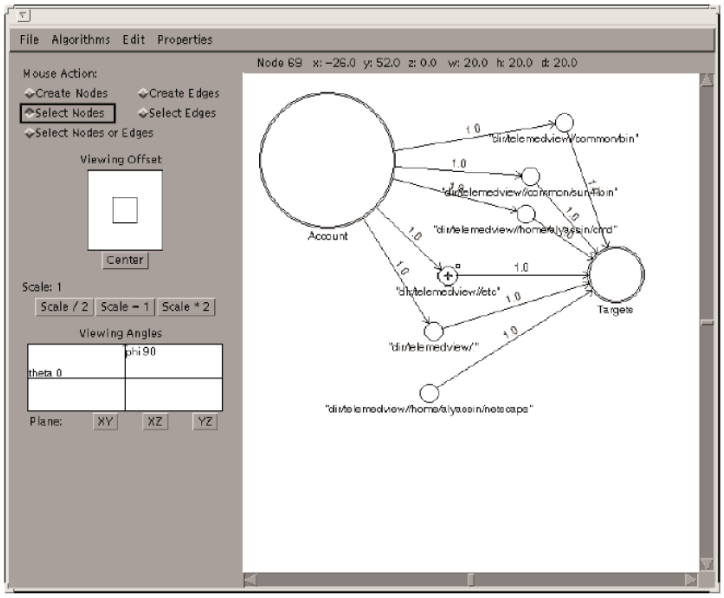

Figure 145. Vector Graph Expanded View.

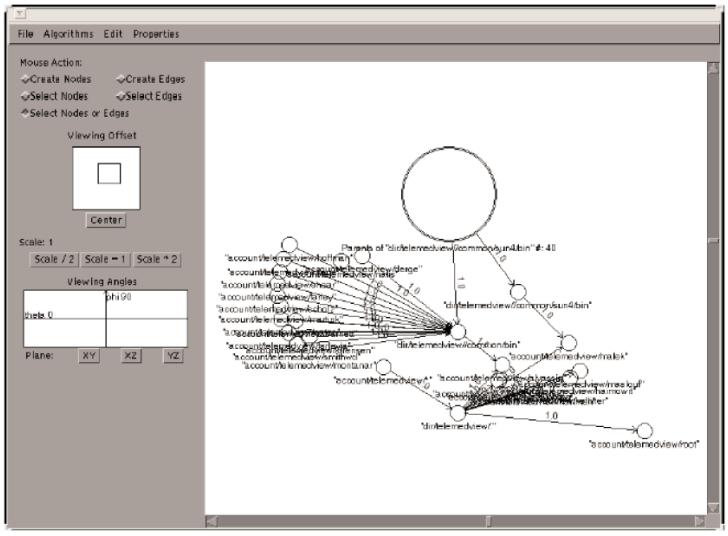

Figure 146. Target Details Expanded.

Security Analysis Models *and finally choos*e Probabilistic Analysis. A text window, shown in Figure 148, should appear which states the probability of successful attack followed by the result graph shown in Figure 149. The result graph shows the most probable path of attack highlighted. The edge values are normalized between zero and one to represent the probability of an attacker choosing that path.

Now let us re-run the analysis using the maximal flow algorithm [12]. *Choose* ***File*** *and* ***Open GML****. Then choose the* ***gml*** *directory and choose the example.gml fil*e. The graph window should appear. *Select* ***Host C Vul n 4*** *again and choose* ***Algorithms*** *and* ***Security Analysis Models*** *and* ***Max Flow Analysis****.* The text window shown in Figure 150 should appear as well as the graph results shown in Figure 151. The edge values have been changed to show the maximum flow along each edge towards the target node. In this case there is a flow of 1.0 and a flow of 5.0 that can reach the target node.

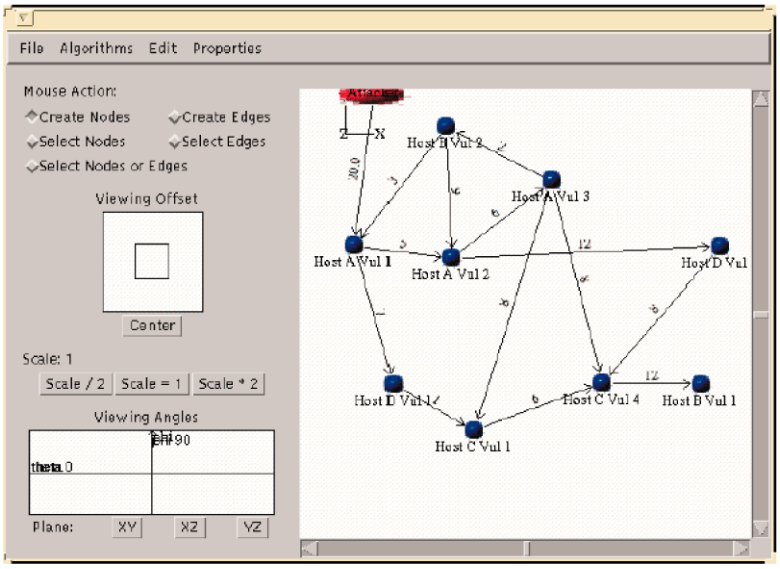

Figure 147. Probabilistic Attack Path Analysis.

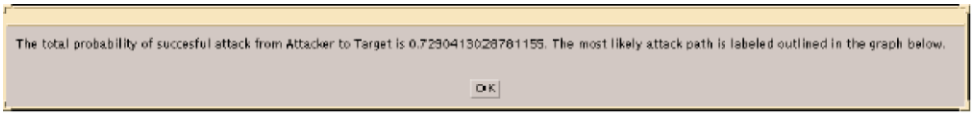

Figure 148. Probability of Attack.

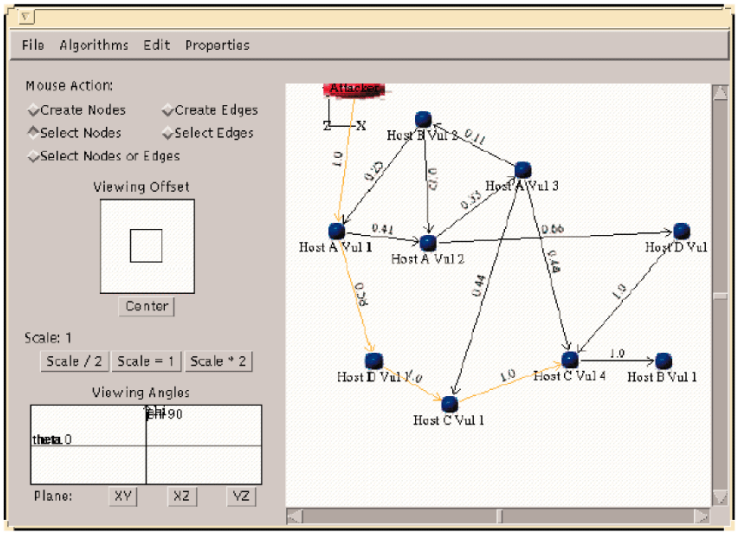

Figure 149. Most Likely Attack Path.

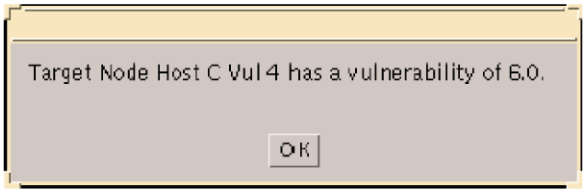

Figure 150. Maximum Flow Results.

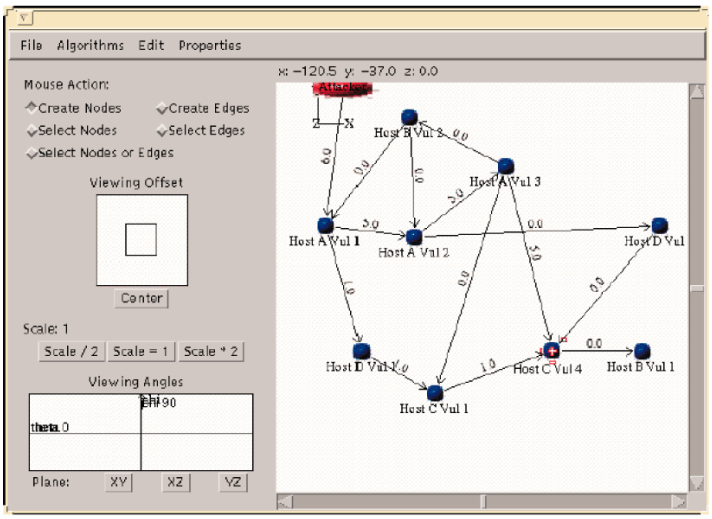

Figure 151. Maximum Flow Graph.

Appendix B
Draft Standard: Inline Network Management

B.1. IN-LINE NETWORK MANAGEMENT PREDICTION DRAFT-IETF-BUSH-INLINE-PREDICTIVE-MGT-00

Status of this Memo

- This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.
- Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet- Drafts.
- Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
- The list of current Internet-Drafts can be accessed at http:// www.ietf.org/ietf/1id-abstracts.txt.
- The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.
- This Internet-Draft will expire on December 30, 2002.
- Copyright Notice: Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

In-line network management prediction exploits fine-grained models of network components, injected into the communication network, to enhance network performance. Accurate and fast prediction of local network state enables more intelligent network control resulting in greater performance and fault tolerance. Accurate and fast prediction requires algorithmic capability. Active and Programmable Networking have enabled algorithmic information to be dynamically injected into the network allowing enhanced capability and flexibility. One of the new capabilities is enhanced network management via in-line management code, that is, management algorithms embedded within intermediate network devices. In-line network management prediction utilizes low-level algorithmic transport capability to implement low-overhead predictive management.

A secondary purpose of this document is to provide general interoperability information for the injection of general purpose algorithmic information into network devices. This document may help in some manner to serve as a temporary bridge between Internet Protocol and Active and Programmable Network applications. This may stimulate some thought as to the content and format of "standards" information potentially required for Active Networking. Management of the Internet Protocol and Active and Programmable Networking is vital. In particular, coexistence and interoperability of active networking and Internet Protocol management is specified in order to implement the injection of algorithmic information into a network.

Implementation Note

This document proposes a standard that assumes the capability of injecting algorithmic information, i.e. executable code, into the network. Active or programmable capability, as demonstrated by recent implementation results from the DARPA Active Network Program, Active Internet Protocol [8] or recent standards in Programmable Networking [9], help meet this requirement. While in-line predictive management could be standardized via a vehicle other than active packets, we choose to use active networking as a convenient implementation for algorithmic change within the network.

B.2. INTRODUCTION

This work in progress describes a mechanism that allows a distributed model, injected into a network, to predict the state of the network. The concept is illustrated in Figure 152. The state to be predicted is modeled within each actual network node. Thus, a distributed model, shown in the top plane, is formed within the actual network, shown in the bottom plane. The top plane slides ahead of wallclock time, although in an asynchronous manner. This means that each simulated node MAY have its own notion of simulation time.

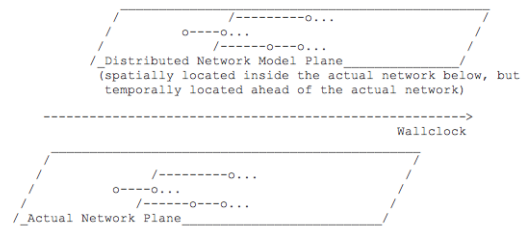

Figure 152. The Distributed Model Inside the Network.

This concept opens up a set of interoperability issues which do not appear to have been fully addressed. How can distributed model components be injected into an existing network? In-line models are injected into the network assuming the overlay environment shown in Figure 153. In-line models in Figure 152 are designed to run as fast as possible in order to maintain a simulation time that is ahead of wallclock, communicating via virtual messages with future timestamps. What if messages are processed out-of-order because they arrive out- of-order at a node? How long do you wait (and slow your simulation down) to make sure they are not out-of-order? This specification provides a framework that allows synchronization to be handled in any manner; e.g. via a conservative (blocking) or optimistic (Time-Warp) manner within the network. Additionally, how can the models verify and maintain a reasonable amount of accuracy? A mechanism is provided in this document to allow local verification of prediction accuracy. Attempts to adjust accuracy are implementation dependent. How do independent model developers allow their models to work coherently in this framework? Model operation is implementation dependent, however, this specification attempts to make certain that model messages will at least be transported in an inter-operable manner, both across and WITHIN, intermediate network devices. How does one publish their model descriptions? How are predicted values represented and accessed? Suggestion solutions for these questions are presented in this document as well.

Overview

In-line predictive network management, which enables greater performance and fault tolerance, is based upon algorithmic information injected into a network allowing system state to be predicted and efficiently propagated throughout the network. This paradigm enables management of the network with continuous projection and refinement of future state in real time. In other words, the models injected into the network allow state to be predicted and propagated throughout the network enabling the network to operate simultaneously in real time and in the future. The state of traffic, security, mobility, health, and other network properties found in typical Simple Network Management Protocol (SNMP) [2]. Management Information Bases (MIB) is available for use by the management system. To enable predictive management of applications, new MIBs will have to be defined that hold both current values as well as values expected to exist in the future.

The AgentX [5] protocol begins to address the issue of independent SNMP agent developers dynamically and seamlessly interconnecting their agents into a single MIB under the control of a master agent. AgentX specifies the protocol between the master and sub-agents allowing the sub-agents to connect to the master agent. The AgentX specification complements this work-in-progress, namely, in-line network management prediction. The in-line network management prediction specification provides the necessary interface between agent functionality injected remotely via an Active Packet and dynamically linked' into a MIB. The agent code may enhance an existing MIB value by allowing it to return predicted values. Otherwise, coexistence with AgentX is SUGGESTED. The in-line network management prediction specification enables faster development of MIB modules with more dynamic algorithmic capability because Active and Programmable networks allow lower-level, secure, dynamic access to network devices. This has allowed injection of predictive capability into selected portions of existing MIBs and into selected portions of active or programmable network devices resulting in greater performance and fault tolerance.

Outline

This document proposes standards for the following aspects of in-line predictive management:

- SNMP Object Time Series Representation and Manipulation
- Common Algorithmic Description
- Multi-Party In-line Predictive Model Access and Control
- Common Framework for Injecting Models into the Network
- Model Interface with the Framework

The high-level components of this proposed standard are shown in Figure 153. The Active Network Framework [10] is a work in progress. In-line Predictive Management is the subject of this document. The Internet Protocol and SNMP are well-known.

Figure 153 shows the various ways in which in-line predictive management can be used in an active network given an implementation in a particular execution environment. The in-line predictive management application runs as an active application on an active node. The framework is independent of the underlying architecture of the active network, which can take one of two forms. The protocol stack on the left shows a fully active network in which the Node Operating System runs one or more Execution Environments. Multiple active applications may execute in any Execution Environment. The protocol stack on the right shows the architecture of an active network overlay over IP. Essentially, the overlay scheme uses the Active Network Encapsulation Protocol (ANEP) [7] as a conduit to use the underlying IP network. The predictive management application executes alongside the other active applications and interacts with any managed active applications to provide their future state. Since the predictive management application requires only the execution environment to run in, it is independent of whether the active network is implemented as an overlay or it is available as a fully active network.

The next section provides basic definitions. Following that, the goals of this proposed standard are

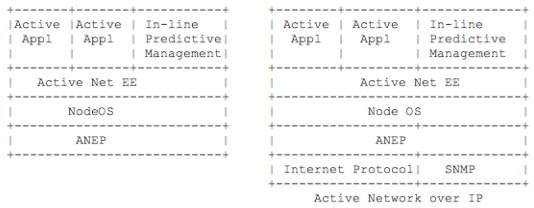

Figure 153. Relationship Among Underlying Assumptions about the Predictive Management Environment.

laid out. The remainder of the document develops into progressively more detail defining interoperability among algorithmic in-line network management prediction components. Specifically, predictive capability requires careful handling of the time dimension. Rather than change the SNMP standard, a tabular technique is suggested. Then, in order to simplify design of predictive management objects, an extension to Case Diagrams is suggested for review and comment. This is followed by the specification of a distributed predictive framework. It is understood that multiple distributed predictive mechanisms exist, however, this framework is presented for comment and review because it contains all the necessary elements. Finally, the detailed interface between the active or programmable code and IP standard interfaces is presented.

Definitions

The following acronyms and definitions are helpful in understanding the general concept of predictive network management.

In-line	Located within, or immediately adjacent to, the flow of network traffic.
Predictive Network Management	The capability of reliably predicting network events or the state of the network at a time greater than wall-clock time.
Fine-Grained Models	Small, light-weight, executable code modules that capture the behavior of a network or application component to enable predictive network management.
Algorithmic Information	Information, in the form of algorithms contained inside executable code, as opposed to static, non-executable data. Depending upon the complexity of the information to be transferred, an algorithmic form, or an optimal tradeoff between algorithmic and non-algorithmic form can be extremely flexible and efficient.
Non-Algorithmic Information	Information that cannot be executed. Generally requires a highly structured protocol to transfer with well-defined code pre- installed at all points in route including source and destination.
Small-State	Information caches that can be created at network nodes, intended for use by executable components of the same application.

Global-State	Information caches created at network nodes, intended to be used by executable components of different applications.
Multi-Party In-line Predictive Management Model	An in-line predictive management model comprised of multiple in- line algorithmic models that are developed, installed, utilized, and administered by multiple domains.

The following acronyms and definitions are useful in understanding the details of the specific predictive network management framework described in this document.

A (Anti-Toggle)	Used to indicate an anti-message. The anti-message is initiated by rollback and is used to keep the system within a specific range of prediction accuracy.
AA (Active Application)	An active network protocol or service that is injected into the network in the form of active packets. The active packets are executed within the EE.
Active Network	A network that allows executable code to be injected into the nodes of the network and allows the code to be executed at the nodes.
Active Packet	The executable code that is injected into the nodes of an active network.
Anti-Message	An exact duplicate of a virtual message except for that the Anti- toggle bit is set. An Anti-message is used to annihilate an invalid virtual message. This is an implementation specific feature relevant to optimistic distributed simulation.
DP (Driving Process)	Generates virtual messages. Generally, the DP is implemented as an algorithm that samples network state and transforms the state into a prediction. The prediction is represented by a virtual message.
EE (Execution Environment)	The active network execution environment. The environment that resides on active network nodes that executes active packets.
Lookahead	The difference between Wallclock and LVT. This value is the distance into the future for which predictions are made.
LP (Logical Process)	An LP consists of the Physical Process and additional data structures and instructions which maintain message order and correct operation as a system executes ahead of real time
LVT (Local Virtual Time)	The LP contains a notion of time local to itself known as LVT. A node's LVT may differ from other nodes' LVT and Wallclock. LVT is a local, asynchronous notion of time.
M (Message)	The message portion of a Virtual Message is implementation specific. This proposed standard SUGGESTS that the message contents be opaque, however, an SNMP varbind, intended to represent future state, MAY be transported. Executable code may also be transported within the message contents.
NodeOS (Node Operating System)	The active network Operating System. The supporting infrastructure on intermediate networks nodes that supports one or more execution environments.
PP (Physical Process)	A PP is an actual process. It usually refers the actual process being modeled, or whose state will be predicted.
QS (Send Queue)	A queue used to hold copies of messages that have been sent by an LP. The messages in the QS may be sent as anti-messages if a rollback occurs.
Rollback	The process of adjusting the accuracy of predictive components due to packets arriving out-of-order or out-of-tolerance. Rollback is specific to optimistic distributed simulation techniques and is thus an implementation specific feature.
RT (Receive Time)	The time message value is predicted to be valid.

RQ (Receive Queue)	A queue used in the algorithm to hold incoming messages to an LP. The messages are stored in the queue in order by receive time.
SQ (State Queue)	The SQ is used as a LP structure to hold saved state information for use in case of a roll-back. The SQ is the cache into which pre-computed results are stored.
Tolerance	A user-specified limit on the amount of prediction error allowed by an LP's prediction.
TR (Real Time)	The current time as a time-stamp within a virtual message.
TS (Send Time)	The LVT that a virtual message has been sent. This value is carried within the header of the message. The TS is used for canceling the effects of false messages.
VM (Virtual Message)	A message, or state, expected to exist in the future.
Wallclock	The current time.

Goals

The goals of this document are...

- *Simplicity*—This document attempts to describe the minimum necessary elements for in-line management prediction. Model developers should be able to inject models into the network allowing SNMP Object value prediction. Such models should work seamlessly with other predictive models in the network. The goal is to minimize the burden on the model developer while also insuring model interoperability.
- *Conformance*—This document attempts conformance with existing standards when and where it is possible to do so. The concept is to facilitate a gradual transition to the active and programmable networking paradigm.
- *In-line Algorithmically-Based Management*—This document attempts to introduce the use of in-line algorithmic management information.

B.3. A COMMON REPRESENTATION OF SNMP OBJECT TIME SERIES FOR IN-LINE NETWORK MANAGEMENT PREDICTION

SNMP, as currently defined, has a very limited notion of time associated with state information. The temporal semantics are expected to be applied to the state by the applications reading the information. On the other hand, predictive management requires generation, handling and transport of information that understands the temporal characteristics of the state, i.e. whether the information is current, future, or perhaps past information. In other words, capability for handling the time dimension of management information needs to be extended and standardized in some manner. In this section, we propose a mechanism for handling time issues in predictive management that require minimal changes from the SNMP standard.

A proposed standard technique for handling the time dimension in predictive state systems is to build the SNMP Object as a Table Object indexed by time. This is shown in the following excerpt from a Load Prediction MIB...

Figure 154. MIB Structure for Handling Object Values with Predictive Capability.

In Figure 155, the result of an SNMP query of the relevant predictive MIB Object is displayed. Because the identifiers are suffixed by time, the object values are sorted temporally. If a client wishes to know the next predicted event on or before a given time, the query can be formulated as a GET-NEXT with the next predicted event time to be determined as the suffix. The GET-NEXT-RESPONSE will contain the next predicted event along with its time of occurrence. Otherwise, a value outside the table will be returned if no such predicted value yet exists.

This allows SNMP GET-NEXT operations from a client to locate an event nearest to the requested time as well as search in temporal order for next predicted events.

B.4. A COMMON ALGORITHMIC DESCRIPTION

SNMP, as currently defined, assumes that non-algorithmic descriptive information will be generated, handled, or transported. Prediction requires model development and execution. This proposed standard SUGGESTS that models are to be small, low-overhead, and fine-grained. Fine-grained refers to the fact that the models are locally constrained in time and space. In this section, we propose algorithmic descriptions of management models designed to encourage the understanding and use of in-line predictive management techniques.

```
.
.
loadPrediction OBJECT IDENTIFIER ::= {  loadPredMIB 1 }

loadPredictionTable OBJECT-TYPE
SYNTAX SEQUENCE OF LoadPredictionEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table of load prediction information."
::= {  loadPrediction 1 }

loadPredictionEntry OBJECT-TYPE
SYNTAX LoadPredictionEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table of Atropos LP prediction information."
INDEX {  loadPredictionPort }
::= {  loadPredictionTable 1 }

LoadPredictionEntry ::= SEQUENCE {
loadPredictionID
DisplayString,
loadPredictionPredictedLoad
INTEGER,
loadPredictionPredictedTime
INTEGER
}

loadPredictionID OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The LP identifier."
::= {  loadPredictionEntry 1 }

loadPredictionPredictedLoad OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the predicted load on the link."
::= {  loadPredictionEntry 2 }

loadPredictionPredictedCPUTime OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the predicted processor time used by a packet
on this node."
::= {  loadPredictionEntry 3 }

loadPredictionPredictedTime OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time at which the predicted event will be valid."
::= {  loadPredictionEntry 4 }
.
.
~
.
.
```

Case Diagrams[4] provide a well-known representation for the relation of management information to information flow as shown in Figure 156. The

```
    ^   Upper Layer
    |
 ==+==  outPackets
    |
    ~
    |
    +==> encodingErrors
    |
    ~
    |
 ==+== inPackets
    ^
    |   Lower Layer
```

Figure 156. An Example Case Diagram.

details of Case Diagrams will not be discussed here (see the previous reference for more information). The purpose of this section is to illustrate an enhancement to the diagram that allows algorithmic information to be specified, particularly for multi-party predictive model interaction.

An excerpt of an SNMP Case Diagram serves to provide a flavor of its current format. The diagram below shows packets arriving from a lower network layer. Some packets are determined to have encoding errors and are discarded. The remaining packets flow to the upper layer.

For the purposes of in-line predictive management, models SHOULD be specified and injected into the system. These models MAY coexist with the current SNMP management model supplementing the information with predictive values. This is denoted by adding algorithmic model information to the Case Diagram. A'+' sign after the name of an

Object Identifier identifies the object as one that can return future values. The model used to predict the future information is written within braces near the Object identifier and incorporates the name of the SNMP object identifiers. This document SUGGESTS using a common syntax for the notation such as that used for code blocks by the C Programming Language block constructs, Java Programming Language blocks, or the notation used by any number of other languages. Standardization of the model syntax is outside the scope of interest for this document. All functions MUST be defined. Operating system function calls MAY NOT be used. The salient point is that the algorithm must be clearly and concisely defined. The algorithm must also be a faithful representation of the actual predictive model injected into the system. As shown in Figure 157,'encodingErrors' is predictively

```
    ^   Upper Layer
    |
 ==+==  outPackets
    |
    ~
    |
    +==> encodingErrors+ {  0.1 * inPackets }
    |
    ~
    |
 ==+== inPackets
    ^
    |  Lower Layer
```

Figure 157. A Sample Algorithmic Description.

enhanced to be 10% of "'inPackets' for future values. The predictive algorithm MUST run on the network node and MUST be immediately available as input for other predictively enhanced objects. The predicted value MUST be available as a response to SNMP queries for future state information, or for transfer to other nodes via virtual messages, explained later in this document. SNMP Objects that are enhanced with predictive capability are

```
.
loadPredictionTable.loadPredictionEntry.loadPredictionID.1 -> OCTET STRING- (ascii):AN-1

loadPredictionTable.loadPredictionEntry.loadPredictionPort.1 -> INTEGER: 3325

loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.4847 -> INTEGER: 240
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.20000 -> INTEGER: 420
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.40000 -> INTEGER: 460
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.60000 -> INTEGER: 497
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.80000 -> INTEGER: 540
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.100000 -> INTEGER: 580
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.120000 -> INTEGER: 619
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedLoad.140000 -> INTEGER: 660

loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.4847 -> INTEGER: 4847
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.20000 -> INTEGER: 20000
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.40000 -> INTEGER: 40000
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.60000 -> INTEGER: 60000
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.80000 -> INTEGER: 80000
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.100000 -> INTEGER: 100000
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.120000 -> INTEGER: 120000
loadPredictionTable.loadPredictionEntry.loadPredictionPredictedTime.140000 -> INTEGER: 140000

loadPredictionTable.loadPredictionEntry.loadPredictionCurrentLoad.1 -> INTEGER: 15949
loadPredictionTable.loadPredictionEntry.loadPredictionCurrentTime.1 -> INTEGER: 25639
.
.
```

Figure 155. Output from a Query of the MIB Structure for Handling Object Values with Predictive Capability.

assumed to always have the actual monitored value at Wallclock time.

If this were a wireless network, a more realistic algorithmic model would likely incorporate channel quality SNMP Objects into the “encodingErrors” prediction algorithm. In many cases, the algorithmic portion of the Case Diagram will involve SNMP objects from other nodes. Syntax should include the ability to identify general topological information in the description of external objects. For example, “inPackets[adj]” or “inPackets[edge]” should indicate immediately adjacent nodes or nodes at the topological edge of the network.

In the example shown in Figure 158, a'packets-

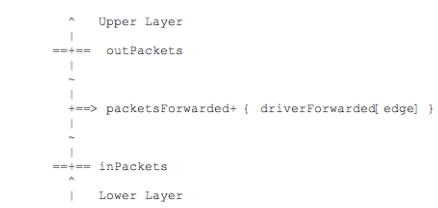

Figure 158. An Algorithmic Description Using State Generated from Another Node Described in Figure 159.

Forwarded' object has predictive capability denoted by the'+' symbol. The predictive capability comes from an algorithmic model specified within the braces next to the object name. In this case, the prediction will be the value of the “driverForwarded” object from the node closest to the edge of the network.

In Figure 159,which is an SNMP diagram of the edge node, the “driverForwarded” object is predicted by executing the algorithm in braces. This

Figure 159. A Node Generating State Information Used by the Node in Figure 158.

algorithm predicts “driverForwarded” packets to be a linear approximation of a sample of “appPackets”. The sample is “epsilon” time units apart and the prediction is “delta” time units into the future.

B.5. MULTI-PARTY MODEL INTERACTION

Multiple developers and administrators of in-line predictive algorithmic models will require mechanisms to ensure correct understanding and operation of each others' models and intentions.

Model Registration

It may be necessary to register predictive models. Registration is often an IANA function [6]. Algorithmic model registration needs to be handled more dynamically than AgentX models. Algorithmic models, while not necessary doing so, have the capability to install / deinstall at rapid rates. The in-line model installation and deinstallation proposed standard is described in Section 7.

Model Interaction

Multiple models residing on a node need to interoperate with one another. This document proposes to use SNMP Object Identifiers as much as possible

for communication of state information among models. In addition, multiple Active Application models may choose to communicate with one another via global state.

Co-existence with Legacy SNMP

Querying an IP addressable node for SNMP objects that are predictively enhanced should appear transparent to the person polling the node. Multiple ports, etc. should not be required. A program injected into a node that serves to extend an SNMP MIB MAY do so using global state. A global state cache holds the SNMP object values and responds via an internal port to connect with a master SNMP agent for the node.

B.6. A COMMON PREDICTIVE FRAMEWORK

This section specifies an algorithmic predictive management framework. The framework allows details of distributed simulation, such as time management, state saving, and model development to be implementation dependent while ensuring in-line interoperability both with, and within, the network. The general predictive network management architecture MUST contain at least one Driving Processes (DP), MAY contain Logical Processes (LP), and MUST use Virtual Messages (VM).

Figure 160 illustrates network nodes containing

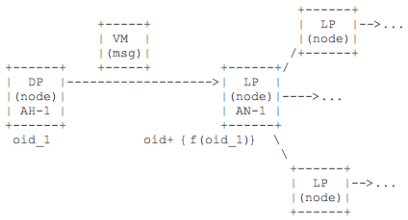

Figure 160. Framework Entity Types.

DPs and LPs. The annotation under nodes AH-1 and AN-1 are an SNMP Object Identifier. SNMP Object Identifier'oid_1' represents state of node AH-1. The predictively enhanced SNMP Object Identifier, “oid+” on node AN-1 is a function of'oid_1'. Note that “f()” is shown as an arbitrary function in the figure, but MUST be well-defined in practice.

The framework makes a distinction between a Physical Process and a Logical Process. A Physical Process is nothing more than an executable task defined by program code i.e. it is the implementation of a particular model or a hardware component or a direct connection to a hardware component representing a device. An example of a Physical Process is the packet forwarding process on a router. Each Physical Process MUST be encapsulated within a Logical Process, labeled LP in Figure 160. A Logical Process consists of a Physical Process, or a model of the Physical Process and additional implementation specific data structures and instructions to maintain message order and correct operation as the system executes ahead of

current (or Wallclock) time as illustrated in greater detail in Figure 160. The details of the DP and LP structure and operation are implementation specific, while the inter-operation of the DP/LP system must be specified. The LP architecture is abstracted in Figure 161. The flow of messages

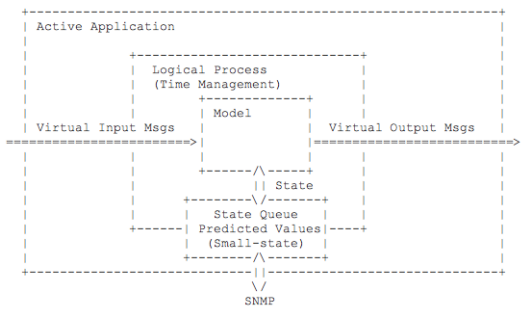

Figure 161. A High-level View of the Logical Process Framework Component within an Active Application.

through the LP is shown by the arrows entering from the left side of the figure. The in-line predictive framework components are shown in Figure 160, where AH-1 and AN-1 are Active Host 1 and Active Node 1 respectively. In this context, active hosts are nodes that can inject new packets into the network while active nodes are nodes that behave as intermediate hops in a network.

The Logical Process MUST handle time management for the model. The Logical Process and the model that it implements MAY be implemented in any manner, however, they must be capable of interoperating. The framework MUST be capable of supporting both conservative and optimistic time management within the network. Conservative time management REQUIRES that the model block when messages MAY be received out-of-order while optimistic time management MAY allow model processing to continue, even when messages are received out-of- order. However, additional implementation specific mechanisms MAY be used to account for out-of-order messages. Such mechanisms MAY be

embedded within the Logical Process and this specification does not attempt to standardize them.

Virtual input messages directed to a Logical Process MUST be received by the Logical Process, passed to the model, and processed. Virtual output messages MAY be generated as a result.

Virtual messages contain the following fields:

- Send Time (TS) which MUST contain the LVT (local simulation time) at which the message was sent
- Receive Time (TR) which MUST denote the time the message is expected to exist in the future
- MAY contain an (optional) Anti-toggle (A) bit for out-of-order message handling purposes such as message cancellation and rollback
- MUST contain the message content itself (M) which is model specific

Thus, a Virtual Message (VM) MUST have the following structure...

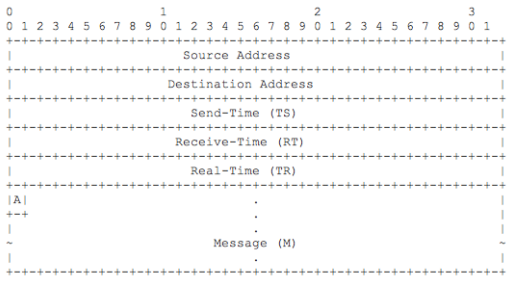

Figure 162. An In-line Management Prediction Virtual Message.

These in-line predictive messages, or virtual messages, that contain invalid fields because the transmitting Logical Processes used an incompatible time management technique MUST be dropped. However, it is SUGGESTED that a count of such packets be maintained in a general in-line predictive management framework MIB. The Receive Time field MUST be filled with the time that this message is predicted to be valid at the destination Logical Process. The Send Time field MUST be filled with the time that this message was sent by the originating

Logical Process. The Anti-Toggle (A) field MUST be used for creating an anti-message to remove the effects of false messages as described later. A message MUST also contain a field for the current Real Time (RT). If a message arrives at a Logical Process out-of-order or with invalid information, that is, out of a pre-specified tolerance for prediction accuracy, it is called a false message. The method for handling false messages is implementation specific. The Receive Queue, shown in Figure 163, maintains

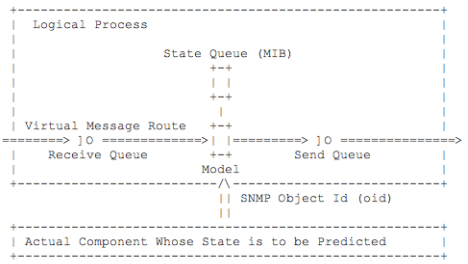

Figure 163. A Logical Process Implementation and Interface.

newly arriving messages in order by Receive Time (TR). The implementation of the Receive Queue is implementation specific.

The Driving and Logical Processes MUST communicate via virtual messages as shown in Figure 164. The Driving Process MAY generate predictions based upon SNMP queries of other layers on the local node. The Logical Process MAY check its prediction accuracy via SNMP queries of other layers on its local node.

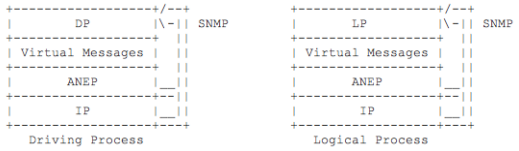

Figure 164. Facility for Checking Accuracy with Actual Network SNMP Objects in the In-line Predictive Management Framework.

The in-line predictive framework MAY allow for prediction refinement and correction by communicating with the actual component whose state is to be predicted via an SNMP query. The asynchronous prediction mechanism has the following architecture for Logical Process (Figure 163).

All of the Logical Process queues and caches MAY reside in an active node's Small-State. Small-State is a persistent memory cache left behind by an active packet that is available to trailing active packets that have the proper access rights. Typically, any type of information can be stored in Small-State.

The Receive Queue MAY maintain active virtual message ordering and scheduling. All active packets

MUST be encapsulated inside Active Packets following the Active Network Encapsulation Protocol [7] format. Once a virtual message leaves the Receive Queue, the virtual time of the Logical Process, known as Local Virtual Time, MUST be updated to the value of the Receive Time from the departing virtual message. Virtual messages MUST originate from Driving Processes, shown in Figure 160 that predict future events and inject them into the system as virtual messages. The development of a Driving Process and Logical Process are dependent upon the model used to enhance the desired state of the system with predictive capability. Logical Processes MUST only operate upon the the arrival of virtual input messages and MUST NEVER spontaneously generate virtual messages.

Following the arrows across Figure 163, virtual messages enter eitherthe Physical Process. The state of the Logical Process is periodically saved in the State Queue (SQ) shown as the State Cache in Figure 163. State Queue values are used to restore the Logical Process to a known safe state when false messages are received. State values are continuously compared with actual values from the Physical Process to check for prediction accuracy, which in the case of load prediction is the number and arrival times of predicted and actual packets received. If the prediction error exceeds a specified tolerance, a rollback MAY occur.

An important part of the architecture for network management is the fact that the State Queue within the in-line management prediction architecture is the node's Management Information Base. The State Queue values are the SNMP Management Information Base Object values; but unlike legacy SNMP values, these values are expected to occur in the future. The State Queue operation is implementation dependent, however, it holds the predicted SNMP Objects, is SUGGESTED to be implemented in small-state, and MUST use the interface specified in Section 7.2 to respond to SNMP queries. The current version of SNMP has no mechanism to indicate that a managed object is reporting its future state; currently all results are reported with a timestamp that contains the current time. In working on predictive active network management prediction there is a need for managed entities to report their state information at times in the future. These times are unknown to the requester. A simple means to request and respond with future time information is to append the future time to all Management Information Base Object Identifiers that are predicted. This requires making these objects members of a Management Information Base table indexed by predicted time as discussed in Section 2. This can be seen in the load Prediction Table shown in Figure 154. Thus a Simple Network Management Protocol client, who does not know the exact time of the next predicted value, can issue a get- next command appending the current time to the known object identifier. The managed object responds with the requested object valid at the closest future time. The figure illustrates an SNMP request and the corresponding response.

Future times are the LVT of the Logical Process running on a particular node. As Wallclock approaches a particular future time, predicted values MAY be adjusted, allowing the prediction to become more accurate. The table of future values MAY be maintained within a sliding Lookahead window, so that old values are removed and the prediction does exceed a given future time. Continuing along the arrows in Figure 161, any virtual messages that are generated as a result of the Physical Process or model computation proceed to the Send Queue (QS).

The Send Queue is implementation dependent, however, it MAY maintain copies of virtual messages to be transmitted in order of their send times. The Send Queue is required for the generation of anti-messages during rollback. Anti-Messages annihilate corresponding virtual messages when they meet to correct for previously sent false messages. Annihilation is simply the removal of both the actual and the anti-message. Where the annihilation occurs is implementation specific and left to the implementor. After leaving the Send Queue, virtual messages travel to their destination Logical Process. Further details on the optimistic synchronization mechanism are implementation dependent and outside the scope of this work in progress.

B.7. SUMMARY OF IN-LINE PREDICTION REQUIREMENTS

An in-line management prediction model developer MUST implement at least one Driving Processing and MAY implement a Logical Process using the same time management technique. The model developer MAY include an SNMP client within the model in order to query the modeled component in order to improve prediction accuracy. The model developer's Driving Process MUST generate virtual

messages. The Logical Process MUST receive and process those messages. The Logical Process MAY respond to virtual messages by generating virtual message(s). The Logical Process MAY use active network node Small- state to hold a time series of the SNMP Object Id whose value is being continuously predicted. The interface to the SNMP MIB small-state is specified in the following section.

B.8. DETAILS OF THE ACTIVE NETWORK INTERFACE

The general active network architectural framework, without any specific network management paradigm implementation, is shown in Figure 165.

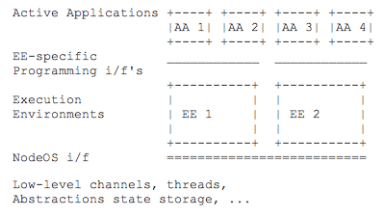

Figure 165. The Active Network Framework.

In-line network management prediction requires a general active network framework that supports active applications to be injected into the proper execution environments. The in-line management prediction framework enforces certain minimal requirements on the execution environment, which are listed below.

Information Caches

The execution environment MUST provide an information cache called 'Small State' as defined in Section 1.3 to enable information exchange between active packets, defined in Section 1.3. The execution environment MAY also provide an information cache called 'Global State', defined in Section 1.3, to enable the in-line management prediction framework to communicate with a predictively managed active application to query its current state. The EE MUST provide an API to be able to store and query both 'Small State' and also to 'Global State', if it is implemented. The EE SHOULD provide appropriate access control mechanisms to both 'Small State' and also to 'Global State', if it is implemented.

Interface to SNMP

The execution environment MUST provide an interface that enables both the in-line management prediction values and the values of the actual component being managed to publish their state to an SNMP MIB. This enables the in-line management prediction framework to store the predicted state in a well-known format and also enables legacy SNMP tools to query the predicted state using SNMP operations. Additionally, the managed application is also able to update its current state using SNMP, which the Logical Process will be able to query. In a particular implementation of such an interface, a generic SNMP agent coded as an active application MAY be injected into the active nodes. The agent creates a 'Global State' on the active node with a well-known name. The agent reads information coded in a known format that has been written to the 'Global State' and publishes it to the MIB. Any active application that wishes to advertise its state uses an interface that enables it to store its information in the well-known 'Global State' in the given format.

The format of the messages that are posted between the SNMP agent and an active application are shown in Figure 166.

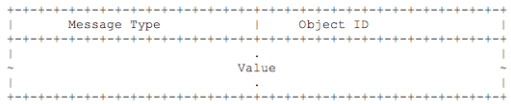

Figure 166. Message Packet.

The SNMP Agent and the active application MAY use special interfaces to implement messaging between them. A Message Packet, whose format is shown in Figure 166, is the basic unit of inter-application communication. Each message consists of a message type. The type SHOULD assume one of the following values:

- MSG_ADDINT: to add a new MIB Object of type SNMP INTEGER
- MSG_UPDATEINT: to update the value of an MIB Object of type SNMP INTEGER
- MSG_GETINT: to get the value of an MIB Object of type SNMP INTEGER
- MSG_ADDLONG: to add a new MIB Object of type SNMP LONG
- MSG_UPDATELONG: to update the value of an MIB Object of type SNMP LONG
- MSG_GETLONG: to get the value of an MIB Object of type SNMP LONG
- MSG_ADDSTRING: to add a new MIB Object of type SNMP STRING
- MSG_GETSTRING: to get the value of an MIB Object of type SNMP STRING
- MSG_UPDATESTRING: to update the value of an MIB Object of type SNMP STRING

The active application SHOULD send a message of the valid message type to the SNMP agent to perform the required operation. On receipt of a message, the SNMP agent SHOULD attempt to perform the requested operation. It MUST then respond with an acknowledgment message in a format shown in Figure 167.

The acknowledgment message has the following format.

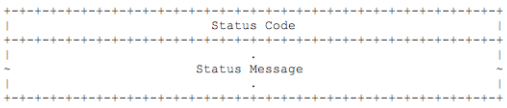

Figure 167. Acknowledgment Message Packet.

The status code MUST have one of the following values:

- OK: to indicate successful operation
- ERR_DUPENTRY: if for a MSG_ADD operation, an Object identifier of given name already exists
- ERR_NOSUCHID: if for a MSG_UPDATE operation, an Object identifier of given name does not exist.

The Status message MAY be any descriptive string explaining the nature of the failure or SHOULD be “Success” for a successful operation.

B.9. IMPLEMENTATION

Models injected into the network allow network state to be predicted and efficiently propagated throughout the active network enabling the network to operate simultaneously in real time as well as project the future state of the network. Network state information, such as load, capacity, security, mobility, faults, and other state information with supporting models, is automatically available for use by the management system with current values and with values expected to exist in the future. In the current version, sample load and processor usage prediction applications have been experimentally validated using the Atropos Toolkit [11]. The toolkit's distributed simulation infrastructure takes advantage of parallel processing within the network, because computation occurs concurrently at all participating active nodes. The network being emulated can be queried in real time to verify the prediction accuracy. Measures such as rollbacks are taken to keep the simulation in line with actual performance.

Predictive In-line Management Information Base

Further details on the in-line network management prediction concept can be found in Active Networks and Active Network Management [1]. The SNMP MIB for the in-line predictive management system described in this proposed standard follows in the next section.

Figure 168. The Atropos MIB. (Printouts appear on the following pages.)

B.10. SECURITY CONSIDERATIONS

Clearly, the power and flexibility to increase performance via the ability to inject algorithmic information also has security implications. Fundamental active network framework security implications will be discussed in [10].

REFERENCES

[B.1] Bush, S. and A. Kulkarni, “Active Networks and Active Network Management (ISBN 0-306-46560-4)”, March 2001.

[B.2] Case, J., Mundy, R., Partain, D. and B. Stewart, “Introduction to Version 3 of the Internet-standard Network Management Framework”, RFC 2570, April 1999.

[B.3] Wijnen, B., Harrington, D. and R. Presuhn, “An Architecture for Describing SNMP Management Frameworks”, RFC 2571, May 1999.

[B.4] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, “Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2)”, RFC 1450, April 1993.

[B.5] Daniele, M., Wijnen, B., Ellison, M. and D. Francisco, “Agent Extensibility (AgentX) Protocol Version 1", RFC 2741, January 2000.

[B.6] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs”, BCP 26, RFC 2434, October 1998.

[B.7] University of Pennsylvania, USC/Information Sciences Institute, University of Pennsylvania, BBN Technologies, University of Pennsylvania, University of Kansas and MIT, “Active Networks Encapsulation Protocol”, July 1997.

[B.8] MIT and MIT, “The Active IP Option”, September 1996.

```
ATROPOS-MIB DEFINITIONS ::= BEGIN

IMPORTS
MODULE-IDENTITY, OBJECT-TYPE, experimental,
Counter32, TimeTicks
FROM SNMPv2-SMI
DisplayString
FROM SNMPv2-TC;

atroposMIB MODULE-IDENTITY
LAST-UPDATED "9801010000Z"
ORGANIZATION "GE CRD"
CONTACT-INFO
"Stephen F. Bush bushsf@crd.ge.com"
DESCRIPTION
"Experimental MIB modules for the Active Virtual Network
Management Prediction (Atropos) system."
::= { experimental active(75) 4 }

--
-- Logical Process Table
--

lP OBJECT IDENTIFIER ::= { atroposMIB 1 }

lPTable OBJECT-TYPE
SYNTAX SEQUENCE OF LPEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table of Atropos LP information."
::= { lP 1 }

lPEntry OBJECT-TYPE
SYNTAX LPEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table of Atropos LP information."
INDEX { lPIndex }
::= { lPTable 1 }

LPEntry ::= SEQUENCE {
lPIndex          INTEGER,
lPID             DisplayString,
lPLVT            INTEGER,
lPQRSize         INTEGER,
lPQSSize         INTEGER,
lPCausalityRollbacks INTEGER,
lPToleranceRollbacks INTEGER,
lPSQSize         INTEGER,
lPTolerance      INTEGER,
lPGVT            INTEGER,
lPLookAhead      INTEGER,
lPGvtUpdate      INTEGER,
lPStepSize       INTEGER,
lPReal           INTEGER,
lPVirtual        INTEGER,
lPNumPkts        INTEGER,
lPNumAnti        INTEGER,
lPPredAcc        DisplayString,
lPPropX          DisplayString,
lPPropY          DisplayString,
lPETask          DisplayString,
lPETrb           DisplayString,
lPVmRate         DisplayString,
lPReRate         DisplayString,
lPSpeedup        DisplayString,
lPLookahead      DisplayString,
lPNumNoState     INTEGER,
lPStatePred      DisplayString,
lPPktPred        DisplayString,
lPTdiff          DisplayString,
lPStateError     DisplayString,
lPUptime         TimeTicks
}

lPIndex OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The LP table index."
::= { lPEntry 1 }

lPID OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The LP identifier."
::= { lPEntry 2 }

lPLVT OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the LP Local Virtual Time."
::= { lPEntry 3 }

lPQRSize OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the LP Receive Queue Size."
::= { lPEntry 4 }

lPQSSize OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the LP send queue size."
::= { lPEntry 5 }

lPCausalityRollbacks OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the number of rollbacks this LP has suffered."
::= { lPEntry 6 }

lPToleranceRollbacks OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the number of rollbacks this LP has suffered."
::= { lPEntry 7 }

lPSQSize OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the LP state queue size."
::= { lPEntry 8 }

lPTolerance OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the allowable deviation between process's
predicted state and the actual state."
::= { lPEntry 9 }

lPGVT  OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is this system's notion of Global Virtual Time."
::= { lPEntry 10 }

lPLookAhead  OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is this system's maximum time into which it can
predict."
::= { lPEntry 11 }

lPGvtUpdate OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the GVT update rate."
::= { lPEntry 12 }

lPStepSize OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the lookahead (Delta) in milliseconds for each
virtual message as generated from the driving process."
::= { lPEntry 13 }

lPReal OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the total number of real messages received."
::= { lPEntry 14 }
```

```
lPVirtual OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the total number of virtual messages
received."
::= { lPEntry 15 }

lPNumPkts OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the total number of all Atropos packets
received."
::= { lPEntry 16 }

lPNumAnti OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the total number of Anti-Messages transmitted
by this Logical Process."
::= { lPEntry 17 }

lPPredAcc OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the prediction accuracy based upon time
weighted average of the difference between predicted and real
values."
::= { lPEntry 18 }

lPPropX OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the proportion of out-of-order messages
received at this Logical Process."
::= { lPEntry 19 }

lPPropY OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the proportion of out-of-tolerance messages
received at this Logical Process."
::= { lPEntry 20 }

lPETask OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the expected task execution wallclock time for this
Logical Process."
::= { lPEntry 21 }

lPETrb OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the expected wallclock time spent performing a
rollback for this Logical Process."
::= { lPEntry 22 }

lPVmRate OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the rate at which virtual messages were
processed by this Logical Process."
::= { lPEntry 23 }

lPReRate OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time until next virtual message."
::= { lPEntry 24 }

lPSpeedup OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the speedup, ratio of virtual time to wallclock time,
of this logical process."
::= { lPEntry 25 }

lPLookahead OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the expected lookahead in milliseconds of this
Logical Process."
::= { lPEntry 26 }

lPNumNoState OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the number of times there was no valid state to
restore when needed by a rollback or when required to check
prediction accuracy."
::= { lPEntry 27 }

lPStatePred OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the cached value of the state at the nearest
time to the current time."
::= { lPEntry 28 }

lPPktPred OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the predicted value in a virtual message."
::= { lPEntry 29 }

lPTdiff OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time difference between a predicted and an
actual value."
::= { lPEntry 30 }

lPStateError OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the difference between the contents of an application
value and the state value as seen within the virtual message."
::= { lPEntry 31 }

lPUptime OBJECT-TYPE
SYNTAX INTEGER (0..2147483647)
--SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is the time in milliseconds that Atropos has been
running on this node."
::= { lPEntry 32 }

END
```

[B.9] IETF, “Proposed IEEE Standard for Application Programming Interfaces for Networks”, October 2000.

[B.10] Princeton University, “Active Network Framework”, July 2002.

[B.11] <http://avnmp.sourceforge.net/download.html>

Glossary and Definitions

Algorithmic Sufficient Statistic	The shortest program S* that computes a finite set S containing d on a universal computer, such that the two-part description consisting of S_ and log|S| is as short as the shortest single program that computes d without input [172].
Complexity Theory	A term used to describe a breadth of disciplines engaged in the study of what makes something hard and what makes something easy. The sense in which something is hard or easy separates the varieties of complexity theory, Kolmogorov complexity considers minimal descriptions less complex than long descriptions. Computational complexity, on the other hand, considers a problem hard if it requires a long time or a lot of space on a Turing machine in order to be solved [211].
Computational Complexity	For input of length n, if Turing machine T makes at most t(n) moves before it stop then T is said to run in time t(n) and have time complexity t(n). If T uses at most s(n) tape cells in the same computation it is said to use s(n) space and have space complexity s(n) [211].
Computational Mechanics	Refers to the structure of a process, in a class of complexity theory sometimes called "structural complexity theory," of which computational mechanics is the most developed [212].
Computational Complexity	The amount of time or memory required to solve a given problem. [211].
Entropy Rate	With respect to a stochastic process, the entropy rate is the rate with which the entropy of a sequence of n random variables grows within [118].
Inductive Inference	The process of reaching a general conclusion from specific examples, including conclusions about examples not specified. Generalization, or reasoning from the specific to the General Case. [213]
Information Assurance	Information operations (IO) that protect and defend information and information systems (IS) by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities. Alternatively, Information operations (IO) that protects and defend information and information systems (IS) by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities. [214]
Kolmogorov Complexity	The length of the smallest program capable of generating a given string without input on a Universal Turing Machine. Sometimes referred to as descriptive complexity or Kolmgorov-Chaitin complexity [10].
Minimum Description Length (MDL)	Criteria for inductive inference [215].
Minimum Message Length (MML)	Criteria for inductive inference [14].
Minimum Sufficient Statistic	A statistic that is a function of all other statistics and contains no additional irrelevant information [118].
Prefix Code	A code in which no code word is the prefix of another codeword such that the it can be instantaneously decoded [118].
Prefix Free Program Set	A set of programs such that no program leading to a halting computation is the prefix of another program [10].

Partial Recursive Functions	The set of functions mapping strings in the set {0, 1}* to the finite set {0,1}* or infinite set {0, 1}∞ that is computable by a Turing Machine [10].
Recursive	A Turing machine that implements a function mapping an input to output and halting on all inputs is known as recursive. All recursive functions are computable [10].
Sophistication	The minimal length of a total recursive function that leads to an optimal two-part code for a given object (binary string). One part is the model that comprises the structure or patterns in the string. The second part consists of random data that identifies the specific object within the set defined by the model. The minimum sufficient statistic in the recursive model class [181].
Statistic	A function of a sample of data [118].
Sufficient Statistic	A statistic of a distribution that contains all the information in a sample about the distribution [118].
Two-part Codes/ Two-part Description	A description of a binary string object consisting of two separate parts. The first part consists of a description of a model or set comprising the compressible parts of the object. The second part consists of the enumeration of the object given the first part and can be considered a description of the random aspects of the object [181].
Typical Element of a Set	An element of a set that can be described most succinctly by an index from an enumeration of all elements of the set. If first describing a sub-set and then enumerating the element can more succinctly describe an element of a set then it is not a typical element [172].