INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

PROBABILISTIC REASONING MOBILE AGENT SYSTEM FOR NETWORK TESTING

by

LAILA KHREISAT

A dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2000

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

4/13/2000
Date

Chair of Examining Committee

4/18/2000
Date

Executive Officer

Prof. Michael Conner

Prof. Myung Lee

Prof. Moughtar Bokli
Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

PROBABILISTIC REASONING MOBILE AGENT SYSTEM FOR NETWORK TESTING

By

Laila Khreisat

Adviser: Professor Tarek Saadawi

Telecommunication networks undergo continuous testing such as trouble shooting, fault isolation, vulnerability assessment, and intrusion detection, to ensure the proper operation of the network. Current testing schemes can sometimes overload the network, with increased bandwidth and resource usage. Networks have a dynamic nature, where the state of the network undergoes constant change, and the available set of tests to perform can be too large to be applied to the network simultaneously. In addition, certain testing schemes such as vulnerability and intrusion detection systems are considered as inefficient [2], lack a generic structured building strategy [12], and are difficult to upgrade [2].

We present an intelligent mobile agent system for testing telecommunication networks. The system is a general-purpose method that can be used for any type of network testing, for example, vulnerability assessment, and intrusion detection. The system consists of mobile agents equipped with tests (indices to tests) to be performed on targets in a network. The tests and the targets are selected using probabilistic

reasoning in a manner that maximizes the probability of selection of problematic nodes in the network. The algorithm takes into consideration the fact that problems such as vulnerabilities and intrusions for example, tend to spread once they have entered the network. Thus once the algorithm detects a problematic node, it selects the most vulnerable nodes within the neighborhood of the selected node and applies the appropriate set of tests to them.

Each mobile agent carries a pointer to the set of tests to be performed on a specific node in the network. The nodes selected for testing constitute a subset of the available nodes in the network. The number of mobile agents deployed and the number of tests per agent are kept within the maximum allowable as dictated by the bandwidth and resource availability of the network. This is crucial in networks were bandwidth and resource limitations prohibit the application of large pools of tests simultaneously. Thus our method selects tests and targets in an optimum manner that ensures detection of any problems within the network in a timely and efficient manner, without overloading the network.

Acknowledgments

I wish to thank my adviser Prof. Tarek Saadawi for his guidance and support throughout the duration of this work. I am also grateful for the support, encouragement and understanding of my husband, and my two children. I also would like to thank my parents for their help and support.

Table of Contents

List of Tables

List of Figures

1. Introduction

Telecommunication networks undergo continuous testing such as trouble shooting, fault isolation, vulnerability assessment, and intrusion detection, to ensure the proper operation of the network. Current testing schemes can sometimes overload the network, with increased bandwidth and resource usage. Networks have a dynamic nature, where the state of the network undergoes constant change, and the available set of tests to perform can be too large to be applied to the network simultaneously.

We present an intelligent mobile agent system for testing telecommunications networks. The system is a general purpose-testing scheme that can be used for any type of network testing, such as vulnerability assessment, and intrusion detection.

Our intelligent mobile agent system uses probabilistic reasoning for test and target selection. The algorithm takes into consideration the fact that problems such as vulnerabilities and intrusions for example, tend to spread once they have entered the network. Thus once the algorithm detects a problematic node, it selects all nodes within the neighborhood of the selected node and applies the appropriate set of tests to them. Although probabilistic reasoning methods are known to suffer from many disadvantages [4,5], such as, the need for complete information, their intractability (need exponential time for execution), and their incompleteness, we propose the use of a new reasoning method developed in [5], that does not suffer from these problems. The method is an adaptive method where the accuracy of results improve gradually as computation time increases, providing a trade-off between resource consumption and output quality. The method consists of three strategies, BASIC, INEQS, and EXPSN. Depending on the time and resource limitations and accuracy of

results needed, either one of these strategies can be used. BASIC is the most efficient strategy, however, it produces wider intervals for the probabilities. INEQS tries to generate tighter intervals than BASIC. EXPSN is the most sophisticated strategy. It tries to compensate for missing information by using a recursive method of substitution. This strategy is more time and resource consuming than the previous two strategies, however, it gives more accurate results.

We generalize the reasoning method developed in [5] to handle two dimensional reasoning [19]. The original reasoning method [11] can only reason with respect to one dimension, meaning that tests are related to the whole network. For example, T_i refers to test number i throughout the whole network. Whereas in the generalized version, i.e. the two dimensional version, tests are denoted by T_{ij}, which refers to test number i on target number j. Thus in the generalized version test and target selection is done simultaneously.

1.1 Previous Work

Previous work has been done on the use of intelligent mobile agents in implementing network security. [15] propose an architecture for active defense of computer systems against intrusions by employing autonomous mobile agents. The mobile agents are trained to detect anomalous activity in the system's traffic by being subjected to a training phase. Agents use genetic programming to actually learn to detect anomalous activity. Genetic programming allows for both feedback learning; human-guided learning; and discovery; finding new combinations of activities to monitor for [14]. However, agent training takes time and is tailored to one specific system that is being

monitored. Intelligent mobile agents for vulnerability detection has been proposed and implemented by [3]. In their scheme, mobile agents are equipped with sets of assessment tests to be applied to nodes (hosts) within the network to detect vulnerabilities. A modified genetic algorithm was used for test selection.

1.2 Mobile Agent Technology

1.2.1 Introduction

Mobile agent technology is seen as the most attractive solution in dealing with the new challenges facing networks today. These challenges include increased bandwidth requirement and network management, resulting from the rapid growth of the internet. The technology allows the implementation of more flexible and decentralized network architectures.

A mobile agent is a self-contained and identifiable computer program that can move within the network and act on behalf of the user or another entity [10]. Mobile agents can meet provided they are at the same location. They can also communicate with one another even if they are at different locations. The main goals for using mobile agents in general, is the reduction of network traffic and asynchronous interaction [10].

Intelligent Mobile agents are viewed as sophisticated software entities possessing artificial intelligence that autonomously travel through a network environment and make complex decisions on behalf of the user.

Mobile agent technology models a network of computers, as a collection of multiple agent-"friendly" environments acting as agent servers by offering a service to mobile

agents that enter, and the agents are modelled as programmatic entities that move from location to location, performing tasks for users [12].

The mobile agent technology has three major components: an agent programming language, an interpreter, and agent protocols [16]. The agent language is used to program the agents and the places. An interpreter for interpreting the language, and agent protocols that allow interpreters residing on different computers to exchange agents.

1.2.2 Current Status of Mobile Agent Systems

The most cited applications for mobile agent systems are: intelligent information retrieval, network and mobility management, electronic commerce, and network services.

More specifically, research has been conducted in the use of intelligent mobile agents in implementing network security. [15] propose an architecture for active defense of computer systems against intrusions by employing autonomous mobile agents. The mobile agents are trained to detect anomalous activity in the system's traffic by being subjected to a training phase. Agents use genetic programming to actually learn to detect anomalous activity. Genetic programming allows for both feedback learning; human-guided learning; and discovery; finding new combinations of activities to monitor for [14]. However, agent training takes time and is tailored to one specific system that is being monitored. Intelligent mobile agents for vulnerability detection has been proposed and implemented by [2]. In their scheme, mobile agents are

equipped with sets of assessment tests to be applied to nodes within the network to detect vulnerabilities. A modified genetic algorithm was used for test selection.

2. The Scheme

2.1 Theoretical Basis

As mentioned previously, our scheme uses adaptive probabilistic reasoning to do test and target selection. The theoretical basis for this method is propositional logic which was introduced in the Artificial Intelligence (AI) community by [6]. We use a simplified variant of the more general framework presented in [7]. We start with a propositional language L whose *formulas* are finitely constructed in the usual way from a denumerable set of *primitive propositions* (atoms), and logical connectives $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation) [8]. A *probabilistic formula* is a statement of the form $a_1 P(\psi_1) + \ldots + a_k P(\psi_k) \geq a$, where k is a positive integer, a's are reals, and ψ's are propositional formulas. For example, $2.0 * P(T_1 \vee T_2) - 7.5 * P(T_3) \geq 3.9$ is a propositional formula. A *probabilistic theory,* is a finite set of probabilistic formulas. A semantics for probabilistic formulas is obtained by considering probabilistic interpretations, that is, probability distributions over the set of all possible worlds obtained by assigning truth-values (either *true* or *false*) to the atoms occurring in the formulas. The probability $P(\psi)$ of a propositional formula ψ in a probabilistic interpretation is the sum of probabilities of the possible worlds in which ψ is true. The probabilistic models of a probabilistic formula are exactly those probabilistic interpretations in which the inequality of the formula holds (that is, *true*). As usual, a probabilistic theory entails a probabilistic formula if and only if the formula is *true* in each model of the theory.

Since probabilistic formulas are linear mappings, each probabilistic theory entails a convex hull of consistent probabilities for each propositional formula. In other words, for any probabilistic theory Γ and for any propositional formula ψ, there is a tightest closed interval $[a,b]$ of reals such that Γ entails $a \leq P(\psi) \leq b$. Given any Γ and ψ, determining the tightest interval $[a,b]$ is the probabilistic reasoning problem. Since the tightest interval $[a,b]$ gives the exact answer, any wider interval $[a',b']$ (where $a' \leq a \leq b \leq b'$) is considered an approximate answer.

For convenience, we will use "clause" to mean "propositional clause", "formula" to mean "probabilistic formula", and "theory" to mean "probabilistic theory.

2.2 Adaptive Probabilistic Reasoning

We restrict our attention to probabilistic theories consisting of linear weight inequalities over propositional clauses [7]. Any given probabilistic theory is converted into a system of linear inequalities [9] that explicitly represent the constraints among the probabilities of propositional clauses present in the theory. Solutions of this linear programming problem provide the probabilities of any propositional clause posed as a query.

In addition to the propositional theory and the query, the user of this reasoning system is allowed to specify a set of propositional clauses, called the *control set*; the clauses in the control set are also used in generating the linear inequalities. For adaptive reasoning, the control set, which is initially set to the clauses in the input theory and query, is gradually expanded by adding new clauses to it. The accuracy of the answer

increases with the increase in the control set, and the exact answer is guaranteed in the limiting case when the control set contains all propositional clauses.

2.2.1 The strategies

Three different strategies are used in generating the linear inequalities. In the first strategy, called BASIC, standard probability axioms are used in generating only equalities over the probabilities of only the clauses in the control set. In the second strategy, called INEQS, clauses that are not in the control set result in the generation of inequalities among the probabilities of the clauses in the control set. In the third strategy, called EXPSN, the clauses missing from the control set are recursively substituted by constraints over clauses in the control set. Note that INEQS and EXPSN generate at least all the constraints that are generated by BASIC.

A very important concept is that of a *child* of a clause. A conjunctive clause is said to be a *child* of any maximal proper conjunctive sub-clause. Two children of a clause are said to be *compatible* if and only if they differ in only one literal, which occurs positively in one and negatively in the other. The children relation is also extended to the *descendant* relation in the usual way. For example, $T_1 \wedge T_2, T_1 \wedge T_3$ are both children of T_1, and $T_1 \wedge T_2$ and $T_1 \wedge \neg T_2$ are compatible children of T_1.

Now we describe the three strategies in detail.

A) Strategy BASIC

In strategy BASIC, three kinds of linear equalities are generated from the clauses in the control set D:

1) For each disjunctive clause $\psi = T_1 \vee ... \vee T_m$ $(m > 1)$ such that D contains ψ, each non-conjunctive descendant of ψ, and $T_1 \wedge ... \wedge T_m$, the following linear equality is generated:

$$P(\psi) = \sum\{P(\varphi) \mid \varphi \text{ is a child of } \psi\} - \sum\{P(\varphi) \mid \varphi \text{ is a grandchild of } \psi\}$$

$$+...+(-1)^{m+1} P(T_1 \wedge ... \wedge T_m)$$

2) For each conjunctive clause $\psi = T_1 \wedge ... \wedge T_m$ (m>1) such that D contains ψ, each non-disjunctive ancestor of ψ, and $T_1 \vee ... \vee T_m$, the following linear equality is generated:

$$P(T_1 \vee ... \vee T_m) = \sum_{i=1}^{m} P(T_i) - \sum\{P(\varphi) \mid \varphi \text{ is a non-disjunctive ancestor of } \psi\}$$

$$+...+(-1)^{m+1} P(T_1 \wedge ... \wedge T_m).$$

3) For each non-disjunctive clause ψ and its compatible children φ and φ' in D, the following linear equality is generated:

$$P(\psi) = P(\varphi) + P(\varphi')$$

B) Strategy INEQS

Strategy INEQS extends the BASIC strategy in the sense that if some descendant of a clause is missing from the control set, then instead of discarding the linear equality altogether, a linear inequality is generated. For a disjunctive clause $\psi = T_1 \vee ... \vee T_m$ (m>1) such that D contains ψ, and some of ψ's descendants, a $\geq$ inequality is

generated if the probability of the missing descendant(s) was to be added if it was in the control set D. Otherwise, a $\leq$ inequality is generated. For atomic and conjunctive clauses a $\geq$ inequality is generated if a child is missing.

C) Strategy EXPSN

EXPSN is the most sophisticated of the three strategies, because it expands missing clauses whenever possible. Again it is based on the BASIC strategy, but if some descendant ψ_i (direct or not) of a clause ψ is missing, it tries to replace it by its expansion, meaning, it tries to generate the linear equality corresponding to the ψ_i and replace ψ_i by its expansion in the original linear equality that is being constructed. The expansion procedure is recursive in the sense that if one or more of ψ_i's descendants are missing, then EXPSN tries to expand these clauses too. If a clause cannot be expanded (because some of its descendants are missing, and cannot be expanded), then the linear equality is not generated. An important restriction on EXPSN is that, when trying to expand a disjunctive clause of size m, and its conjunctive descendant ψ_m of size m is missing, EXPSN does not try to expand ψ_m as this would result in an infinite loop.

In all three strategies the basic constraints between probabilities of clauses have to hold. For a two literal case, these constraints are:

$P(T_1 \vee T_2) \geq \{P(T_1), P(T_2)\}$, $P(T_1) \geq P(T_1 \wedge T_2)$, $P(T_2) \geq P(T_1 \wedge T_2)$.

In other words, the probability of a child clause is always less than or equal to the probability of the parent clause, this extends to the *descendant* relation in the usual way.

Let Z denote the probabilistic theory, C the control set and Q the query.

To better understand the method, the following example will go through the cases where we have complete information and the cases of missing information to see what linear inequalities are generated by the three strategies.

We will start with a probabilistic theory Z_1 where some information is missing.

$$Z_1 : P(T_1) = 0.02, \quad P(T_2) = 0.01.$$

Suppose the system is asked to determine the probability of the query $Q_1 = T_1 \vee T_2$. The initial control set (C_1) consists of only the clauses present in the theory and the query, thus C_1 consists of $\{T_1, T_2, T_1 \vee T_2\}$.

BASIC will not generate any equalities since the clause $T_1 \wedge T_2$ is missing. The same is true for EXPSN, since there are no clauses in the control set that can be used to substitute for the missing clause. So both BASIC and EXPSN provide the answer [0.02,1], this answer comes from the fact that $P(T_1 \vee T_2) \geq P(T_1)$, and $P(T_1 \vee T_2) \geq P(T_2)$.

As for INEQS, the following inequality is generated $P(T_1 \vee T_2) \leq P(T_1) + P(T_2)$, and the answer is a tighter interval [0.02,0.03].

For the control set C_1' obtained by adding $T_1 \wedge T_2$ to C_1, all three strategies generate the following equality: $P(T_1 \vee T_2) = P(T_1) + P(T_2) - P(T_1 \wedge T_2)$, and give the answer [0.02,0.03].

If the two clauses $T_1 \wedge \neg T_2$, and $\neg T_1 \wedge T_2$ are added to $C_1^{'}$, which now consists of $\{T_1, T_2, T_1 \vee T_2, T_1 \wedge \neg T_2, \neg T_1 \wedge T_2, T_1 \wedge T_2\}$, then BASIC generates the following equalities:

$$P(T_1 \vee T_2) = P(T_1) + P(T_2) - P(T_1 \wedge T_2)$$

$$P(T_1) = P(T_1 \wedge T_2) + P(T_1 \wedge \neg T_2)$$

$$P(T_2) = P(T_1 \wedge T_2) + P(\neg T_1 \wedge T_2)$$

And this is the case of complete information, no clause is missing from the control set. [1]

If the control set consists of the following clauses: $T_1, T_1 \wedge T_2, \neg T_1 \wedge T_2, T_1 \vee T_2$, then the inequalities/equalities generated are:

EXPSN: $P(T_1 \vee T_2) = P(T_1) + (P(T_1 \wedge T_2) + P(\neg T_1 \wedge T_2)) - P(T_1 \wedge T_2)$

INEQS: $P(T_1) \geq P(T_1 \wedge T_2)$, $P(T_1 \vee T_2) \geq P(T_1) - P(T_1 \wedge T_2)$

BASIC: does not generate any equalities.

Our method of reasoning runs in a time that is polynomial in the size of the control set D [5]. After the constraints (equalities/inequalities) have been generated that capture the probabilistic dependencies among the clauses in the control set, they are combined with those in Z to form a linear programming problem, which is then solved to provide probabilities of arbitrary clauses. Solving a linear programming

---

[1] For the 2 literal case, the set of all clauses is :
$\{T_1, T_2, T_1 \vee T_2, T_1 \vee \neg T_2, \neg T_1 \vee T_2, \neg T_1 \vee \neg T_2, T_1 \wedge T_2,$
$T_1 \wedge \neg T_2, \neg T_1 \wedge T_2, \neg T_1 \wedge \neg T_2\}$. This set can be reduced in half, first by removing complementary clauses ($\neg\psi$ is a complementary clause for ψ) and counterpart clauses ($a \vee \neg b$ is

problem in known to be $O(m^{3.5}E^2)$ [9], where m is the size of the control set (which is equivalent to the number of variables in the corresponding linear program) and E is the sum of the lengths of the constraint set, which is equal to $(3 * Size(D) - 1 + Size(Z)) * (Size(D) + f(n))$, where $f(n)$ is a polynomial in n of degree 3.

2.3. Generalization of the Strategies to the Two Dimensional Case

The probabilistic reasoning method discussed above only handles the one dimensional case, where each atom T_i denotes test number i. In the two dimensional case , each test is denoted by T_{ij}, representing test number i executed on node (host) number j. Assume that there are N nodes in the network. Let $P(host_j)$ denote the probability of selection for node number j. Each atom T_i in the equalities/inequalities generated by the above three strategies will now be replaced by T_{ij}.

Thus for strategy **BASIC** the generalized linear equalities generated are:

1) For each disjunctive clause $\psi = T_{1j} \vee ... \vee T_{mj}$ $(m > 1, j = 1,..., N)$ such that D contains ψ, each non-conjunctive descendant of ψ, and $T_{1j} \wedge ... \wedge T_{mj}$, the following linear equality is generated:

$$P(\psi) = \sum \{P(\varphi) \mid \varphi \text{ is a child of } \psi\} - \sum \{P(\varphi) \mid \varphi \text{ is a grandchild of } \psi\}$$

---

a counterpart of $\neg a \vee b$). Thus the set is reduced to:
$\{T_1, T_2, T_1 \vee T_2, T_1 \wedge \neg T_2, \neg T_1 \wedge T_2, T_1 \wedge T_2\}$.

$$+\ldots+(-1)^{m+1} P(T_{1j} \wedge \ldots \wedge T_{mj})$$

At most N such linear equalities will be generated, one for each node in the network.

2) For each conjunctive clause $\psi = T_{1j} \wedge \ldots \wedge T_{mj}$ $(m > 1, j = 1,\ldots, N)$ such that D contains ψ, each non-disjunctive ancestor of ψ, and $T_{1j} \vee \ldots \vee T_{mj}$, the following linear equality is generated:

$$P(T_{1j} \vee \ldots \vee T_{mj}) = \sum_{i=1}^{m} P(T_{ij}) - \sum \{P(\varphi) \mid \varphi \text{ is a non-disjunctive ancestor of } \psi\} + \ldots + (-1)^{m+1} P(T_{1j} \wedge \ldots \wedge T_{mj}).$$

Again as for case 1, at most N such linear equalities will be generated, one for each node in the network.

3) For each non-disjunctive clause $\psi = T_{1j} \wedge \ldots \wedge T_{mj}$, $(m > 1, j = 1,\ldots, N)$ and its compatible children φ and φ' in D, the following linear equality is generated:

$$P(\psi) = P(\varphi) + P(\varphi')$$

Strategy INEQS

Strategy INEQS extends the BASIC strategy in the sense that if some descendant of a clause is missing from the control set, then instead of discarding the linear equality altogether, a linear inequality is generated. For a disjunctive clause $\psi = T_{1j} \vee \ldots \vee T_{mj}$ (m>1,j=1,...,N) such that D contains ψ, and some of ψ's descendants, a $\geq$ inequality is generated if the probability of the missing descendant(s) was to be added if it was in the control set D. Otherwise, a $\leq$ inequality is generated. For atomic and conjunctive clauses a $\geq$ inequality is generated if a child is missing.

Strategy EXPSN

EXPSN is the most sophisticated of the three strategies, because it expands missing clauses whenever possible. Again it is based on the BASIC strategy, but if some descendant ψ_i (direct or not) of a clause ψ is missing, it tries to replace it by its expansion, meaning, it tries to generate the linear equality corresponding to the ψ_i and replace ψ_i by its expansion in the original linear equality that is being constructed. The expansion procedure is recursive in the sense that if one or more of ψ_i's descendants are missing, then EXPSN tries to expand these clauses too. If a clause cannot be expanded (because some of its descendants are missing, and cannot be expanded), then the linear equality is not generated. An important restriction on EXPSN is that, when trying to expand a disjunctive clause of size m, and its conjunctive descendant ψ_m of size m is missing, EXPSN does not try to expand ψ_m as this would result in an infinite loop.

In all three strategies the basic constraints between probabilities of clauses have to hold. For a network of two nodes and two tests these constraints are:

$P(T_{11} \vee T_{21}) \geq \{P(T_{11}), P(T_{21})\}$, $P(T_{11}) \geq P(T_{11} \wedge T_{21})$, $P(T_{21}) \geq P(T_{11} \wedge T_{21})$,

$P(T_{12} \vee T_{22}) \geq \{P(T_{12}), P(T_{22})\}$ $P(T_{22}) \geq P(T_{12} \wedge T_{22})$, $P(T_{12}) \geq P(T_{12} \wedge T_{22})$.

To find out the probabilities of the individual and combination tests throughout the whole network independent of the hosts, the following equalities are used:

$$P(T_i) = \sum_{j=1}^{N} P(T_{ij}).P(host_j) \quad \text{.....(2.3.1)}$$

where T_i is an atomic clause representing test number i. Equation (2.3.1) gives the probability of test number i being positive throughout the whole network.

For any combination test $\psi = T_1 \wedge ... \wedge T_m$, $(m > 1)$,

$$P(\psi) = \sum_{k=1}^{N} P(T_{1k} \wedge ... \wedge T_{mk}).P(host_k) \quad \text{.....(2.3.2)}$$

for all k=1...N.

And for $\psi = T_1 \vee ... \vee T_m$, $(m > 1)$,

$$P(\psi) = \sum_{k=1}^{N} P(T_{1k} \vee ... \vee T_{mk}).P(host_k) \quad \text{.....(2.3.3)}$$

for all k=1...N.

The one dimensional case is easily derived from the two dimensional case. In the one dimensional case $P(T_{ij}) = P(T_i)$, $P(T_{1k} \vee ... \vee T_{mk}) = P(T_1 \vee ... \vee T_m)$, and $P(T_{1k} \wedge ... \wedge T_{mk}) = P(T_1 \wedge ... \wedge T_m)$.

Using equation (2.3.1),

$$P(T_i) = \sum_{j=1}^{N} P(T_{ij}).P(host_j) = \sum_{j=1}^{N} P(T_i).P(host_j) = P(T_i) \sum_{j=1}^{N} P(host_j) = P(T_i)$$ since

$$\sum_{j=1}^{N} P(host_j) = 1.$$

Using equation (2.3.2),

$$P(T_1 \wedge ... \wedge T_m) = \sum_{k=1}^{N} P(T_{1k} \wedge ... \wedge T_{mk}).P(host_k) = \sum_{k=1}^{N} P(T_1 \wedge ... \wedge T_m).P(host_k)$$

$$= P(T_1 \wedge ... \wedge T_m).\sum_{k=1}^{N} P(host_k) = P(T_1 \wedge ... \wedge T_m)$$

Similarly for the case of disjunctive clauses using equation (2.3.3).

2.4 Objectives and Assumptions

There exists a large pool of tests to be used for testing a network environment . The available pool of tests is too large to be applied all at once due to bandwidth and resource limitations. So the objective of the scheme is to optimize the selection process of both the tests to be performed and the nodes to be tested, in a way that maximizes the probability of selection.

Tests will be denoted by T_{ij} (an atom), representing test number i executed on node number j. Initially, we assume that the probability that any test T_{ij} is positive is $P(T_{ij}) \in [0,1]$, since no information is available. As testing is done $P(T_{ij})$ can be estimated as the relative frequency of the positive occurrences of the test among all tests performed. Although we start out with a fixed set of tests, more tests can be added on as they become available.

2.5 Components of the scheme

The scheme employs an entity called an **Adaptive Assessor** (AA) which consists of a **Reasoning Agent Generator (RAG)** and an **Adaptive Probabilistic Reasoning System (APRS). RAG** consists of two entities: **Agent_Generator and Dispatcher**. **RAG** is responsible for generating agents equipped with tests and dispatching them to targets in the network. The agents perform the specified tests on the targets and record which tests were positive and which ones were negative. This information is reported back to the **Agent_Generator** in **RAG**. Using this information **Agent_Generator** will decide which targets to test and what tests to perform on these

targets the next time around. This is accomplished by constructing a Probabilistic theory **Z** from the information received from the agents. The probabilistic theory Z is then passed on to **APRS** which converts it into a linear program, which is then solved. Targets and their corresponding tests are selected from three different groups. The first group consists of the set of tests that came back positive during the previous stage. The maximum probability, $P(T_{ij})$, is selected, which indicates that test i on node j has the maximum probability among the positive tests. Thus node j will be tested using test number i during the next stage. The second group is the set of new tests that have not been executed yet, selection from this group is done at random. Finally, the last group is the set of negative tests. This selection process ensures that no tests are left out, thus preventing any problems within the network from being undetected. This is crucial, since a negative test may become positive at a later point in time.

2.6 Advanced Two Dimensional Case.

If a vulnerability or an intrusion has been detected at a node in a network, then the likelihood that the neighboring nodes are also vulnerable or have suffered an intrusion is very high. Thus we need to test not just the single node that was selected but also all the nodes lying within the neighborhood of that node. This increases the probability of detection and allows for quicker measures to be taken to prevent any possible damage from happening. We define the neighborhood of a node as a cluster of nodes within which the node is located. All nodes in the same neighborhood must be reachable from each other. This is similar to the first level cluster defined in the

scheme of clustering that is used for grouping network nodes into clusters for hierarchical routing, see [17]. In clustering the set of nodes in the network are divided into groups called first level clusters. First level clusters are grouped into second level clusters and so on until the m-1 level clusters are formed. Where cluster number m is the union of all the m-1 clusters and encompasses all the nodes in the network. All nodes in the same first level cluster must be reachable from each other. This concept of clustering is used for hierarchical routing, and results in smaller routing tables. In this context of network testing we are only using the concept of the cluster for grouping the nodes together. We are not requiring any change to existing routing schemes that are currently being used.

Thus the neighborhood of a node is the first level cluster within which the node is located. Thus if a vulnerable node is detected within a first level cluster, then the G most vulnerable nodes within that cluster will be tested. The task of clustering in this context is abstract in the sense that it merely assigns cluster numbers to the network nodes and can be done by the network administrator [18]. Other variations on the neighborhood of a node can also be defined, for example, the neighborhood could be defined as the subset of nodes that are one hop away from that node, or two hops away. We choose to use the first level cluster as the neighborhood.

Now we are going to illustrate the concept of the neighborhood by an example.

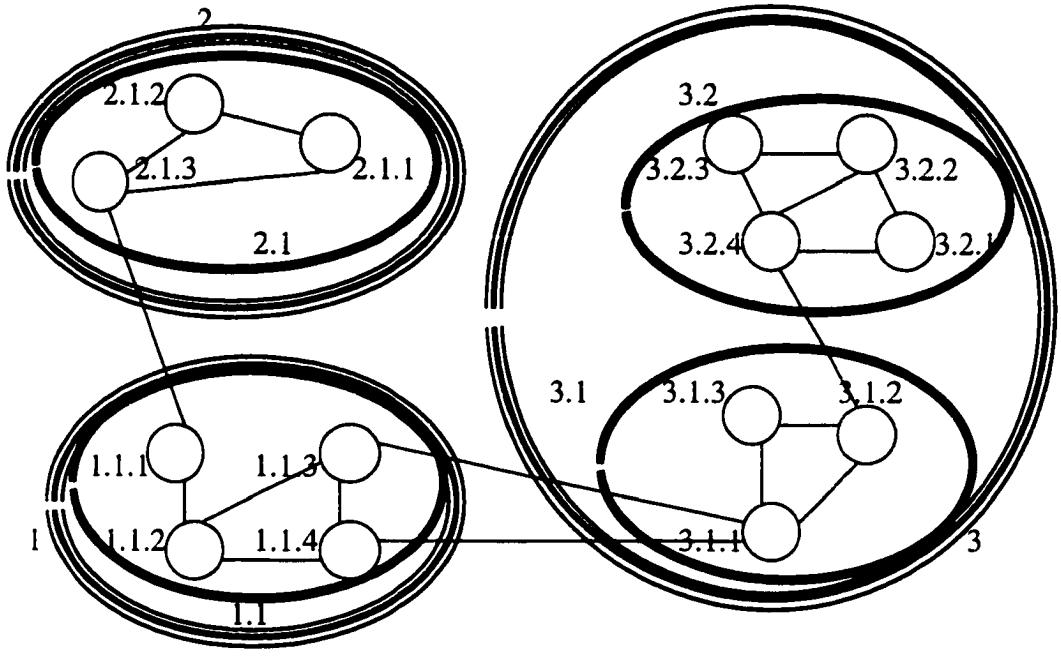

Figure 1. Clustering of network nodes.

Figure 1. shows a network of 14 nodes that has been divided into neighborhoods or clusters. There are four neighborhoods, namely, clusters 1.1, 2.1, 3.1, 3.2. Thus for example, if node 3.2.3 was selected as a vulnerable node and nodes 3.2.2 and 3.2.1 were the most vulnerable nodes within the neighborhood of node 3.2.3, then the nodes selected for testing, are 3.2.1, 3.2.2, 3.2.3.

The advantages of including the neighborhoods of vulnerable nodes in the selection process is that the total number of nodes selected for testing in each stage increases. If the number of vulnerable nodes selected at each stage is denoted by V then using the advanced two dimensional scheme VG vulnerable nodes are selected during each stage, assuming that the G most vulnerable neighbors are selected, compared to only V vulnerable nodes in the one dimensional scheme.

Thus we enhance [19] the two dimensional case to take into consideration neighborhoods of possible vulnerable nodes rather than just single vulnerable nodes as is done in the original two dimensional case. The main idea is that once a vulnerable node has been identified in the network the algorithm proceeds to select the most vulnerable nodes that are within the neighborhood of the selected vulnerable node.

2.6.1 Components of the advanced scheme

The advanced two dimensional scheme uses the same components as the original two dimensional scheme, and the same functionality for each component except for **RAG.** We define a function called neighborhood(k) which returns the neighborhood of node k. **RAG** is modified such that the test and target selection for the set of positive tests is modified to include the neighborhood of vulnerable nodes.

Targets and their corresponding tests are selected from three different groups.

1) The first group consists of the set of tests that came back positive during the previous stage.

   Select $T_{ij} \cup \{T_{ik}, \forall k = 1,..., j-1, j+1,..., h; node_k \in vul(j) \subset neighborhood(j)\}$ such that $P(T_{ij})$ is maximum $\forall i, j; i = 1,..., No\_tests; j = 1,..., h$. vul(j) is the set of the G most vulnerable neighbors of node j

   In other words, for every selected T_{ij} such that $P(T_{ij})$ is maximum, select the G most vulnerable nodes in the neighborhood of node j, where the neighborhood of node j is the set $\{T_{ik}, \forall k = 1,..., j-1, j+1,..., h; node_k \in neighborhood(j)\}$.

Thus for every selected vulnerable node (node j) the G most vulnerable nodes in the neighborhood of node j are also selected for testing.

2) The second group is the set of new tests that have not been executed yet, selection from this group is done at random.
3) Finally, the last group is the set of negative tests.

This selection process ensures that no tests are left out, thus preventing any problems within the network from being undetected. This is crucial, since a negative test may become positive at a later point in time.

3. The Algorithm

In the initial phase of the algorithm no information is available about the relative frequencies of the assessment tests, in other words, the probability that a particular test T_{ij} is positive is unknown. The only thing we can assume is that it is between 0 and 1, i.e., $P(T_{ij}) \in [0,1]$. Therefore, RAG simply selects the tests, and hence the targets, at random. Although the selection during this stage is done at random, the number of agents generated and targets selected is kept within the maximum allowable which is determined by the bandwidth and resource limitations imposed.

After the execution of the initial stage the agents report back their findings to **RAG**. Specifically, each agent will report back which tests were positive indicating the existence of a problem, and which tests were negative. Using this information **RAG** will now decide which targets to test and retest during the next stage, and which combinations of tests to perform on each target. This is accomplished by formulating a probabilistic theory, which is passed on to **APRS**, which performs the adaptive reasoning to obtain the probabilities of the positive tests. This information will be passed back to **RAG** which uses it in deciding the targets to test, and the best combination of tests to perform, for the next stage of execution.

3.1 The Implementation

We have implemented the one dimensional case [11] and the advanced two dimensional cases [19] to study the performance of the testing scheme.

3.1.1 The one dimensional case.

We begin with the one dimensional case. The following is a detailed description of the algorithm.

Algorithm Vul-Assess-1dim():
Inputs:
/* These values are determined from system constraints, namely, available bandwidth and computational resources */

A_MAX: maximum number of agents that can be deployed at the same time.

Q: number of stages of testing to perform.
Variables:
T: list of targets selected.
A: list of agents generated.
P-Pos_tests: list of the probabilities of the positive tests.
P-Neg_Tests: list of the probabilities of the negative tests.
New-Tests: a list of tests not performed yet.
Step 1: Initial step
T = Select-Target(random);
A = Agent-Generator(T , random);
Dispatch-Agent(A);
Collect-Info();
Estiamte-Prob();
Creat-Prob-Theory();
APRS(); /*adaptive probabilistic reasoning*/
Q=Q-1;
Step2:
Repeat
T = Select-Target(smart); /*selects targets in a smart manner*/
A = Agent-Generator(T , non-random) ;
Dispatch-Agent(A);
Collect-Info();
Estimate-Prob();

/* During a fixed interval, the known probabilities do not change */
If No. agents deployed>=0.5*A_MAX Then
Begin
/*Update Prob-Theory */
Create-Prob-Theory();
APRS();
End
Q=Q-1;
Until Q <= 0 /* repeat the above steps Q times */

end(Vul-Assess-1dim).

The algorithm uses several procedures that are briefly described below.

Select-Target(method) is a procedure that selects the targets to be tested according to **method**. If **method** is random, then the targets are selected at random. If **method** is smart, then the targets will be selected from three distinct groups. The group of hosts that tested positive during the previous stage, the group of hosts that tested negative, and finally the group of hosts that have not been tested yet.

Dispatch-Agent() is a procedure that sends an agent, that has been created by **Agent-Generator** to test the selected target(s). The agent performs the selected tests on the target(s) to which it was dispatched. The actual testing was simulated by generating a random number between 0 and 1. If the generated number is less than or equal to 0.5 then the test result is positive, otherwise the result is negative. Another distribution was tried, where the probability of a test being positive was 90%, and the probability of being negative was 10%. The results obtained using this distribution were the same as for the previous distribution. So the first distribution was used for deciding whether a test result is positive or negative.

The **collect()** procedure collects information from the agents. Specifically, for every test performed, it records whether it was positive or negative.

The **Estimate-Prob()** procedure, computes the relative frequencies of the tests and combinations of tests performed.

Create-Prob-Theory() simply creates a list of the probabilities of the tests and test combinations that are known thus far.

Now, we will provide a more detailed description of **Generate-Agent().**

```
Procedure Generate-Agent(T , method)
/* For each target in T, it selects a list of tests according to method. If method is
random, then the tests are selected at random. If method is non-random, then
the tests are selected according to their probabilities. After selection process is
done, create agents to carry out these tests.*/

Begin /* Agent-Generator */
If (method = 'random') Then
Begin
    Select-tests(random);
End
If (method = 'non-random') Then
Begin
    Repeat
        /* for each target in T create a list of tests*/
         For each target in T find the following:
         Begin
            Maximum{P-Pos-Tests);     /* positive test with maximum
                                                    probability*/
            Maximum(P-Neg-Tests);     /* negative test with maximum
                                                    probability */
            Select-random(New-Tests); /* select at random from the set of tests
                                                    never done before */
         End
    Until all targets in T have been considered.
End
End. /* Agent-Generator */
```

To describe the algorithm we will first start with a simple example. Assume an agent was dispatched to test a specific target. The agent had four tests to perform, namely, T_1, T_2, T_3, T_4. Assume that T_1, T_2, and T_3 came back positive while T_4 was negative, and assume the following values for the probabilities were known at that time: $P(T_1)=0.42, P(T_2)=0.35,$ $P(T_3)=0.4,$ $P(T_5)=0.55,$ $P(T_6)=0.38,$ $P(T_1 \cap T_2)=0.3$, where T_5 and T_6 are two new tests, not executed before on this target.

At this point **RAG** constructs a probabilistic theory, which is basically a list of the available probabilities. For this example, the probabilistic theory is the following:

$P(T_1) = 0.42$ $\quad\quad$ $P(T_5) = 0.55$

$P(T_2) = 0.35$ $\quad\quad$ $P(T_6) = 0.38$

$P(T_3) = 0.4$ $\quad\quad$ $P(T_1 \cap T_2) = 0.3$

Notice that $P(T_4)$ and $P(T_1 \cap T_3)$ and $P(T_2 \cap T_3)$ are not known and are initially estimated as [0,1], this means that T_4 ,i.e. test 4, has not tested positive before this, or has come back positive but the updating of the probabilities has not been done yet. Similarly for the other two. At this point **RAG** passes on the probabilistic theory to **APRS**, which performs probabilistic reasoning to obtain the unknown probabilities. The following are the results obtained:

$P(T_1 \cap T_3) = [0,0.4]$, $P(T_2 \cap T_3) = [0,0.35]$, $P(T_4) \in [0,1]$.

RAG will now have to decide the best combination of tests to perform on the particular target the next time around. This is accomplished by finding out the following:

1. The maximum of $\{ P(T_1), P(T_2), P(T_3) \}$ which is $P(T_1)$=0.42.
2. Select at random from the set of new tests, in this example T_5 and T_6 are two new tests that have not been executed. A random number generator is used to generate a random number between 0 and M (total number of tests available). This random number is used to choose between T_5 and T_6. Assume that T_5 is chosen.
3. The maximum of the probabilities of all the negative tests, in this case $P(T_4)$.

According to the above calculations the combination of tests for the next stage will consist of the following: T_1, T_4, T_5.

3.2 The advanced two dimensional case

Now we describe the algorithm for the advanced two dimensional case. Figure 2 below shows a flowchart diagram of the algorithm. The algorithm executes Q stages, however, in the actual implementation of the algorithm the execution continues until a steady state is reached, see section 5 for a description of the steady state.

Algorithm Vul-Assess-2dim():
Inputs:
/* These values are determined from system constraints, namely, available bandwidth and computational resources */

A_MX: maximum number of agents that can be deployed at the same time.
Q: number of stages of testing to perform.
Variables:
A: list of agents generated. An agent consists of a list of tests to perform, and the target on which to perform the tests on.
/* The following 3 arrays, are 2 dimensional arrays, where the row index specifies the test number, and the column index specifies the target (host).
P-Pos_tests: list of the probabilities of the positive tests for the whole network.
P-Neg_Tests: list of the probabilities of the negative tests for the whole network.
New-Tests: a list of tests not performed yet for the whole network.
Step 1: Initial step
A = Agent-Generator(random);
Dispatch-Agent(A);
Collect-Info();
Estiamte-Prob();
Creat-Prob-Theory();
APRS(); /*probabilistic reasoning*/
Q=Q-1;
Step2:
Repeat

```
    A = Agent-Generator(non-random) ;
    Dispatch-Agent(A);
    Collect-Info();
    Estimate-Prob();

    /* During a fixed interval, the known probabilities do not change */
    If No. agents deployed>=0.5*A_MAX Then
    Begin
     /*Update Prob-Theory */
       Create-Prob-Theory();

       APRS();
    End
    Q=Q-1;
Until Q <= 0                /* repeat the above steps Q times */
end(Vul-Assess-2dim).
```

The algorithm uses several procedures that are briefly described below.

Dispatch-Agent() is a procedure that sends an agent, that has been created by **Agent-Generator** to test the selected target(s). The agent performs the selected tests on the target(s) to which it was dispatched. The actual testing was simulated by generating a random number between 0 and 1. If the generated number is less than or equal to 0.5 then the test result is positive, otherwise the result is negative.

The **collect()** procedure collects information from the agents. Specifically, for every test performed, it records whether it was positive or negative.

The **Estimate-Prob()** procedure, computes the relative frequencies of the tests and combinations of tests performed. Where the relative frequency of a test T_{ij} is defined as, $P(T_{ij})$ = (No. positive occurrences of T_{ij}) /Total No. tests .

Create-Prob-Theory() simply creates a list of the probabilities of the tests and test combinations that are known thus far.

Now, we will provide a more detailed description of **Agent-Generator().** This procedure selects from three groups of tests. The first group, **P-Pos-Tests,** is a 2 dimensional array, where the row index denotes the test number, and the column index denotes the target. This array specifies the probabilities of all positive tests throughout the whole network (i.e. for all the nodes (targets)). Each element specifies the probability of positive test i on host j, T_{ij}, for all i<=maximum number of tests, and for all j<=the number of nodes.

The second group, **P-Neg-Tests,** is the same as **P-Pos-Tests**, except it is an array of the probabilities of the negative tests, similarly, **New-Tests**, is an array of the probabilities of the tests never done before,

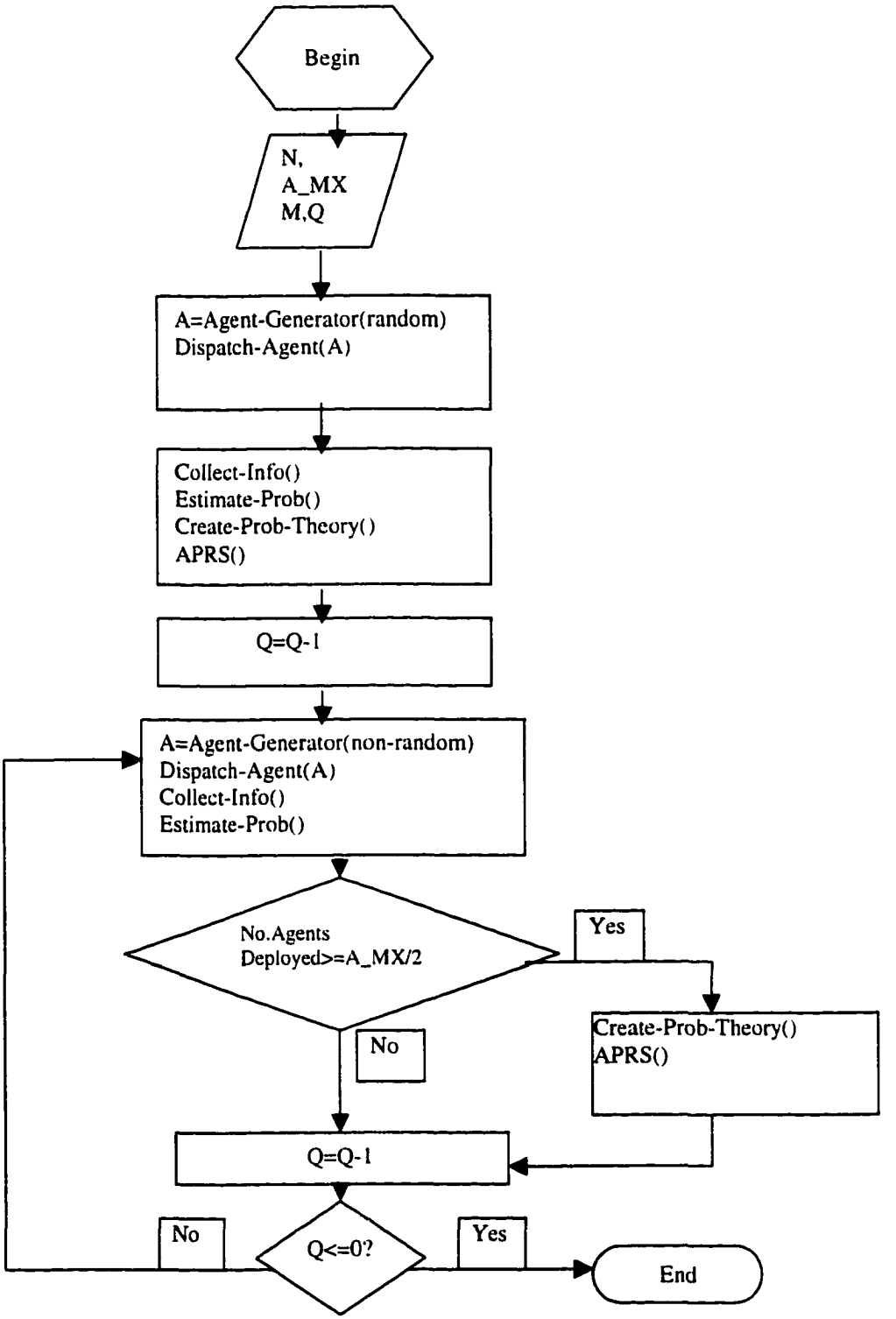

Figure 2 Flowchart Diagram of the Advanced Two Dimensional algorithm.

```
Procedure Agent-Generator(method)
/* Selects tests and targets according to method. If method is random, then the
selection is done at random. If method is non-random, then the tests are selected
according to their probabilities. */

Begin /* Agent-Generator */
If (method = 'random') Then
Begin
    Select-tests(random);
End
If (method = 'non-random') Then
Begin
        PT=Maximum(P-Pos-Tests);      /* positive test with maximum
                                          probability*/
        Vul(Neighborhood(PT),G);      /*Select from neighborhood of PT*/
        Maximum(P-Neg-Tests);     /* negative test with maximum
                                          probability */
        Select-random(New-Tests); /* select at random from the set of tests
                                     never done before */
End
End. /* Agent-Generator */
```

So this procedure selects a set of tests, where each test T_{ij} selected, denotes test number i to be performed on host (node) j.

To describe the algorithm we will first start with a simple example. Assume a 7 node network, and two tests to be performed on the network. Let T_{ij} denote test number i on node j. Assume that the following tests came back positive from the previous stage of execution: $T_{11}, T_{21}, T_{22}, T_{17}, T_{26}, T_{25}, T_{15}$, and that the negative tests were: $T_{12}, T_{23}, T_{13}, T_{24}$, and tests T_{14}, T_{16}, T_{27} have never been performed. Assume the following probabilities are known at this point:

$P(T_{11})=0.42, P(T_{21})=0.35, P(T_{13})=0.4, P(T_{23})=0.5, P(T_{12})=0.37, P(T_{23})=0.7, P(T_{14})=$ 0.4, $P(T_{22})=0.65$, $P(T_{17})=0.19$, $P(T_{25})=0.7$, $P(T_{26})=0.48$, $P(T_{11} \cap T_{21})=0.3$, $P(T_{23} \cap T_{13})=0.4$, $P(T_{15})=05$.

Using these values **RAG (i.e. Agent-Generator)** creates a probabilistic theory that is passed on to **APRS** which performs probabilistic reasoning to obtain the unknown probabilities. The following are the results obtained: $P(T_{15} \cap T_{25})=[0,0.5]$.

RAG will now decide the targets to be tested and what tests to be performed on these targets. This is accomplished by finding out the following:

1. Maximum of $\{P(T_{11}), P(T_{21}), P(T_{22}), P(T_{17}), P(T_{26}), P(T_{25}), P(T_{15})\}$ which is $P(T_{25})=0.7$. Now the most vulnerable nodes in the neighborhood of node 5 are also selected. The neighborhood of node 5 consists of two nodes, namely, nodes 6 and 4, and node 6 is the most vulnerable. Thus at this point two nodes have been selected: nodes 5,6
2. Maximum of negative tests which is $P(T_{23})=0.5$.
3. Select at random from the set of new tests, assume T_{27} is selected.

According to these calculations nodes 5,6 are to be tested using test number 2, and nodes 3 and 7 are to be tested using test number 2.

4. Probabilities of the Tests

As mentioned above, initially, the probability of any test T_i being positive is unknown and is assumed to be $P(T_i) \in [0,1]$. As testing is performed and new values for the probability estimates are obtained, we may end up with single valued probabilities and interval-valued probabilities. So the question is, how do we choose the maximum of these values as is required in the procedure **Generate-Agent().** If all the probabilities are single-valued, then it is simply straightforward, just find the maximum value of these probabilities. If the probabilities are all interval-valued, then to find the maximum we have to consider three cases. This is illustrated by way of an example. Assume that we have two intervals, $[a,b]$ and $[c,d]$.

Case 1: if interval $[a,b]$ is a subset of interval $[c,d]$, i.e. $a >= c, b <= d$ then choose the tightest interval as the maximum, namely, $[a,b]$.

Case 2: if interval $[a,b]$ is not a subset of interval $[c,d]$, then choose the maximum as the interval with the largest values for its bounds. Namely, if $a > c$, and a>d, and $b > d$, then the maximum is the interval $[a,b]$, otherwise, the maximum is $[c,d]$.

Case 3: if intervals $[a,b]$ and $[c,d]$ are overlapping, i.e. $a < c < b$ and $b < d$, then the maximum is $[c,d]$.

In the mixed case where we have both single-valued probabilities and interval-valued probabilities, we treat the single-valued probabilities as an interval with the same upper and lower bound, namely, [a , a], and apply the above cases.

5. Results

5.1 For the one dimensional case

We tested the algorithm assuming a network of 10 nodes to be tested using 30 different tests. We define the steady state of the algorithm as the state when the probabilities of the tests are stable. A test is stable if the probability of the test from the previous stage and the probability at the current stage are within epsilon of each other. In other words $P_k(T_i) - P_{k-1}(T_i) \leq \epsilon$. The probability of miss (not selecting) for nodes was used as a measure of the algorithms' performance. Other measures were also used to study the performance of the algorithm. The ratio of the probability of node selection to the vulnerability of the node. Where probability of node selection was measured as the number of times the node was selected to the total number of node selections done. Node vulnerability is a measure of whether a node suffers from the problem that we are currently testing for using the pool of tests available, and it is measured as the ratio of the total number of positive tests for the node, to the total number of positive tests executed in the network. Another measure of performance was the ratio of the probability of selection for the tests to the probability of a test being positive.

The probability of miss is the probability of failing to select a vulnerable node, and is defined by the following formula: Prob(miss)=1-Prob(Selection|Vulnerable), where Prob(selection|vulnerable) is the probability of selecting a node given that it is vulnerable. In the cases, where no vulnerabilities exist, and when the number of selected nodes (max_h) is equal to the total number of nodes (h) to be tested, the

Prob(miss) should be 0. In the case of random selection of the nodes with no regard to the vulnerability of the nodes in the selection process, the probability of miss is given by $1-\frac{max\_h}{h}$. For our method of selection, we define the probability of miss by the following formula:

$1-\frac{1}{q}\sum_{k=1}^{q} P_k(host(j) \mid Pos(j)\ is\ maximum\ \ \forall i = 0,....,No\_tests, \forall j = 0,....,h)$, where q is the number of stages executed, h is the number of nodes to be tested, max_h is the number of nodes selected for testing at each stage, $P_k(host(j))$ is the probability of selection for node j at stage k, $Pos(j)$ is the number of positive tests at node j, and No_tests is the total number of tests to be performed. The probability of miss for different values of max_h is shown in table 1.

Max_h	Prob(miss)
4	0.36
6	0.24
8	0.21

Table 1. Probability of miss

These values were obtained under steady state conditions. It is apparent that increasing the number of nodes tested simultaneously (max_h) decreases the probability of miss.

Figures 3,4,5 show the plot of the probability of selection versus the vulnerability. Figure 3. shows the plot of the probability of selection versus vulnerability for nodes 0,1,2, and 3. It is apparent from the plot that the ratio of the probability of selection

and vulnerability converges to 1.00 at the steady state. This is expected since the vulnerability of a node determines it's probability of selection, in other words the more vulnerable the node is the more likely it will be selected.

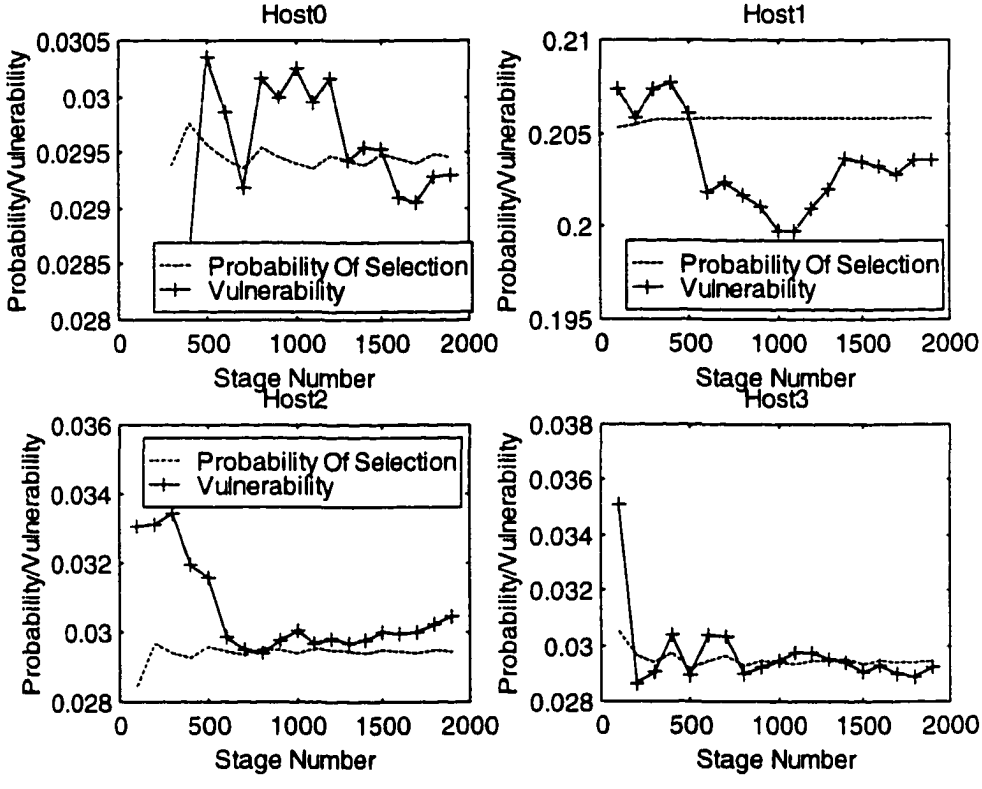

Figure 3. Probability of selection versus vulnerability

The same is true for the other nodes, see figures 4 and 5.

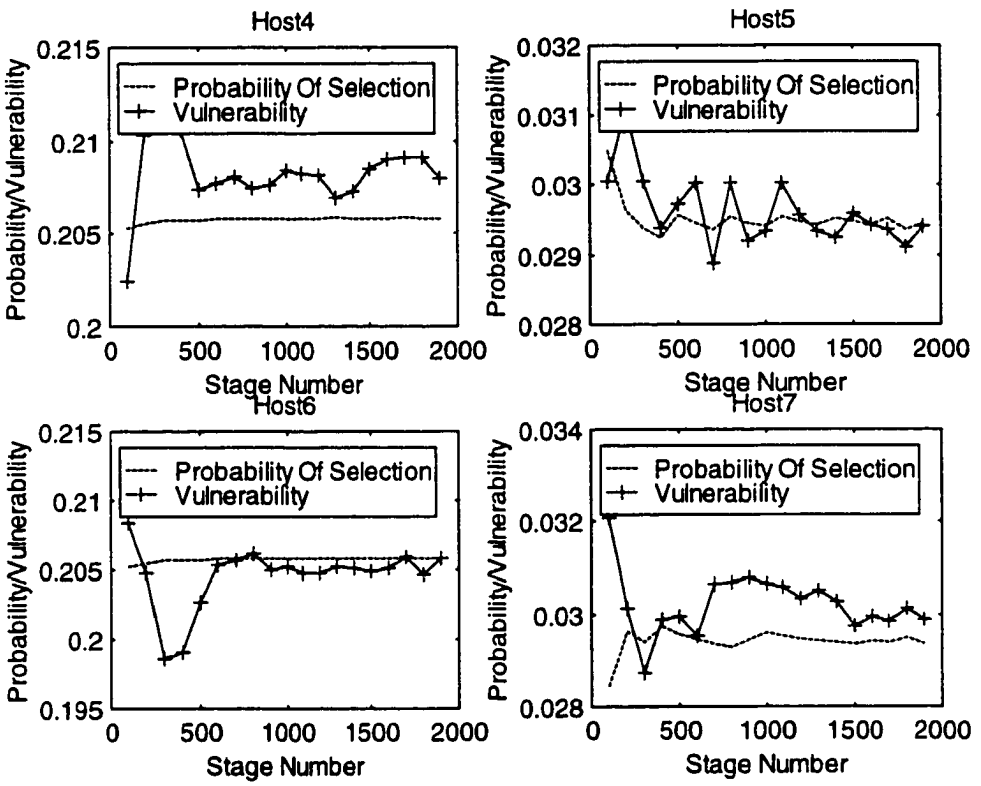

Figure 4. Probability of selection versus vulnerability

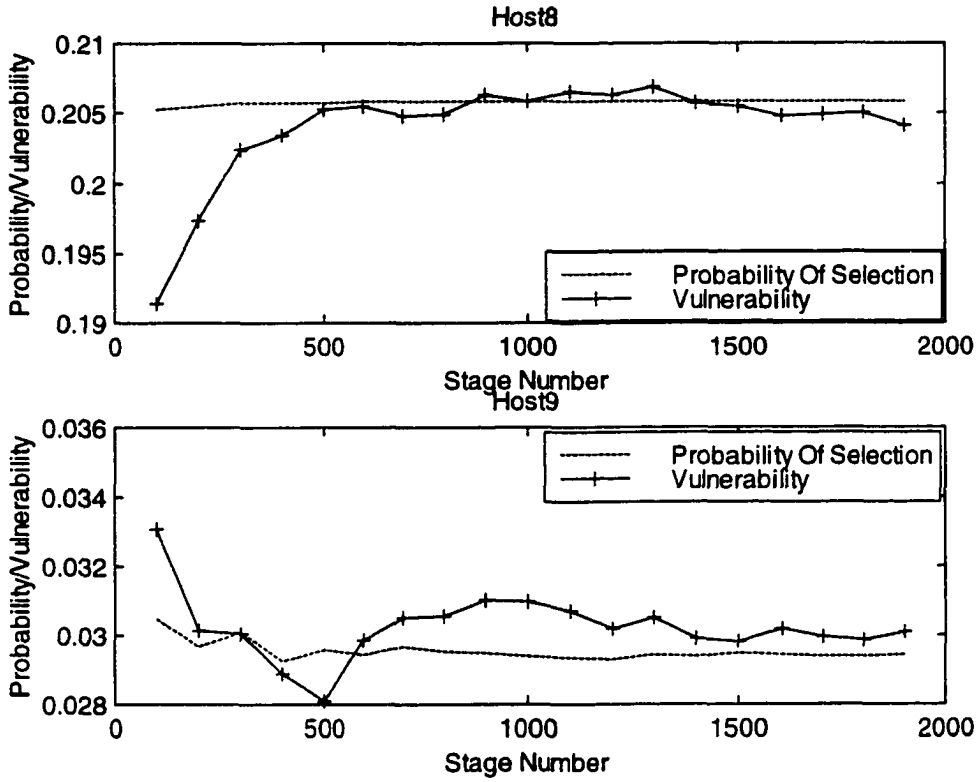

Figure 5. Probability of selection versus vulnerability

Figures 6 and 7 show the plot of the probability of selection versus the probability of being positive for a set of tests for nodes 0 and 1. Comparing the ratio of the probability of selection and the probability of being positive for nodes 0 and 1, it is apparent that for node1 , which is much more vulnerable than node 0, the ratio for tests7 and 29 is almost 1.0, whereas for node 0 the ratio is around 0.1. As for test 1 the ratio is less than 0.1 for both nodes 0 and 1, which is an indication that test 1 is infrequently selected for testing on both nodes.

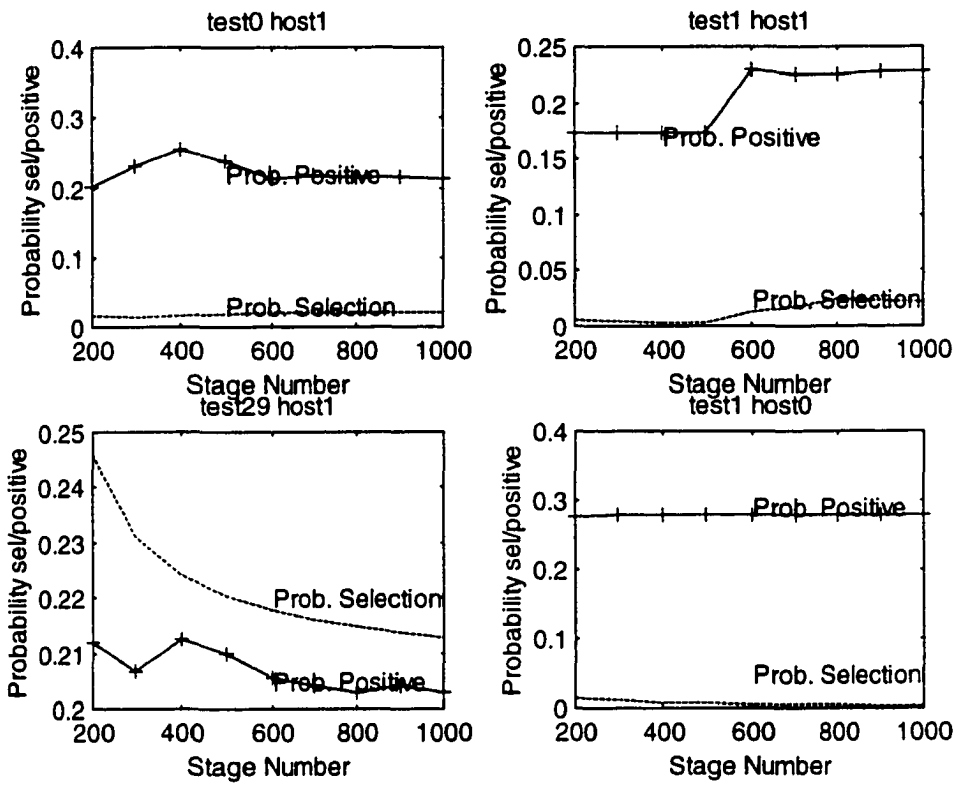

Figure 6. Probability of selection versus probability of being positive

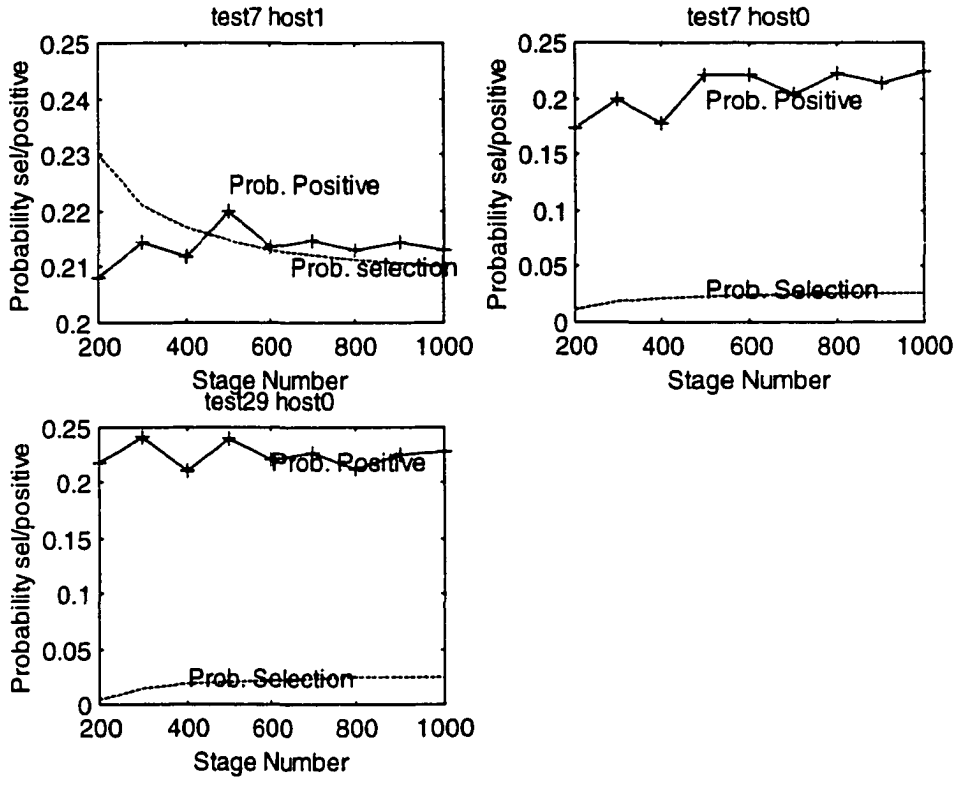

Figure 7. Probability of selection versus probability of being positive

Figure 8 is a plot of the number of times test1 was selected as a negative test and the number of times it was selected as a positive test for all the nodes. It is apparent that test1 is not selected as frequently as test29, see figure 9. In figure 9 the same plot for test 29 is shown.

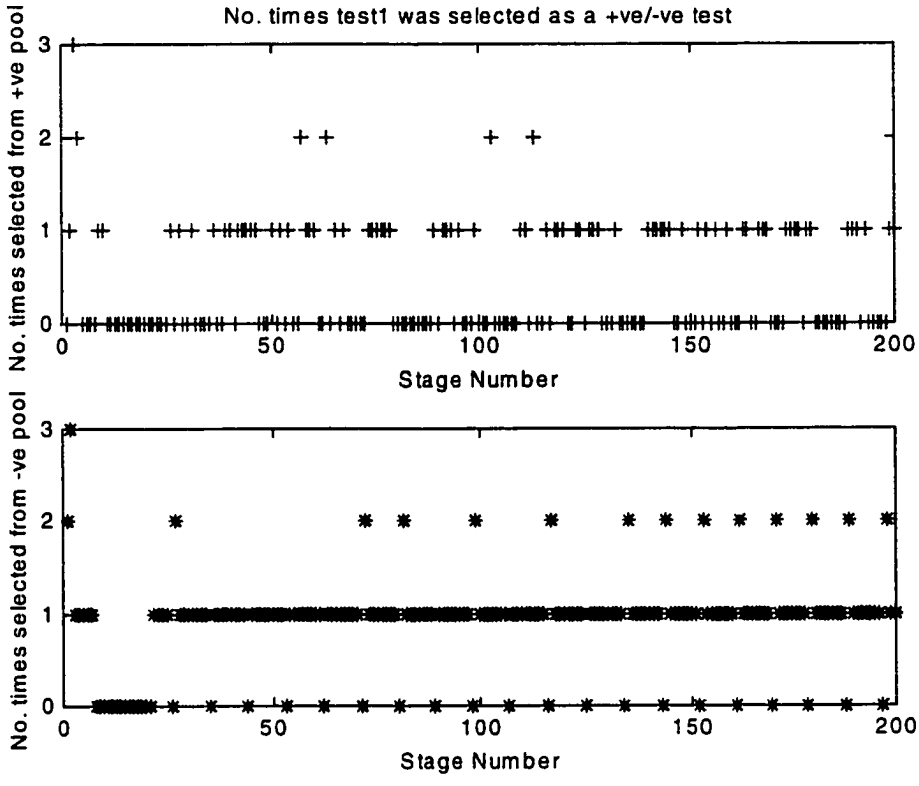

Figure 8.

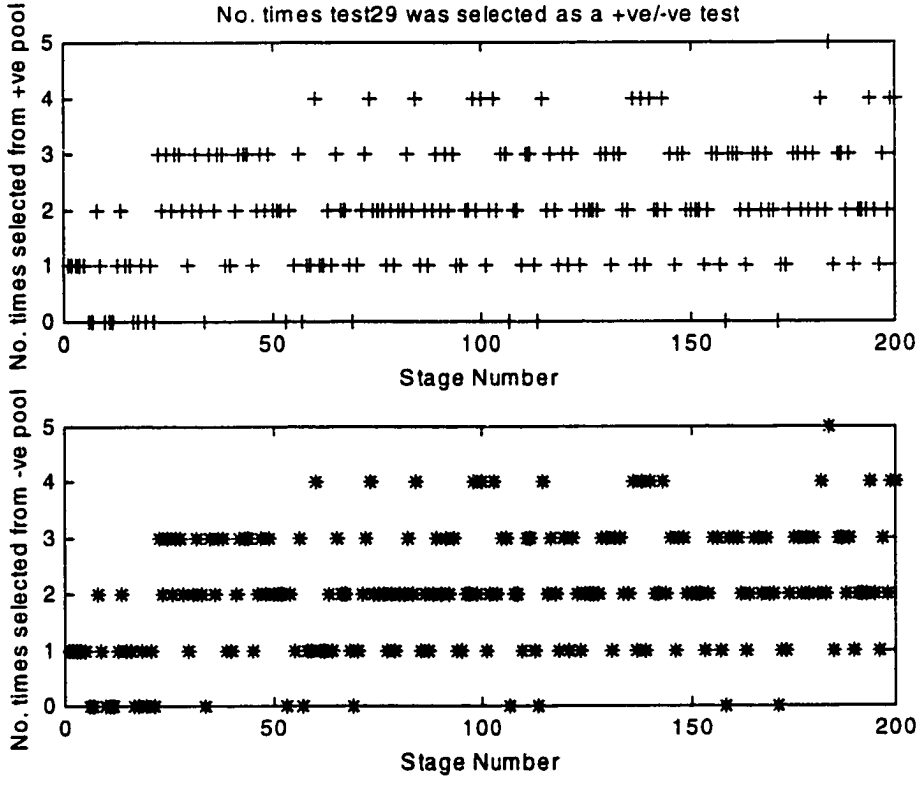

Figure 9.

Figures 10 and 11.are plots of the cumulative total of the number of times a set of tests were selected within 200 stages of execution of the algorithm. Tests 7 and 29 are the most selected tests, up to almost 800 times within the 200 stages. Test 4 is selected up to 250 times, which is then followed by tests 24 and 10, they are selected up to 166 times. The rest of the test set falls between 90 and 24 times.

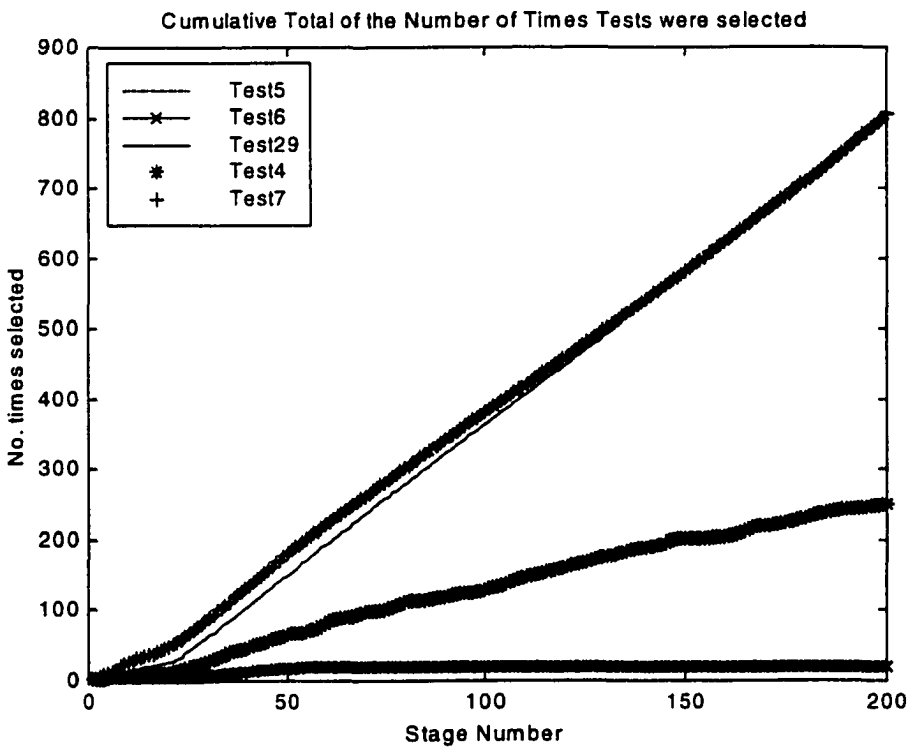

Figure 10.

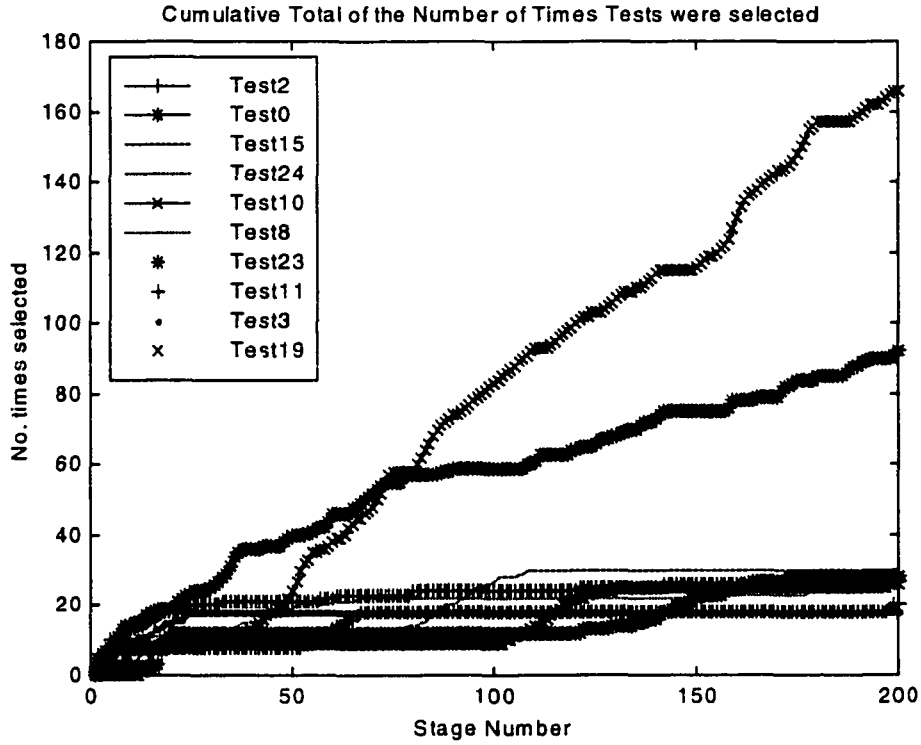

Figure 11.

5.2 The Advanced two dimensional case

For the advanced two dimensional case we assumed a network of 10 nodes. We define the neighborhood of a node to be the first level cluster of nodes, which is the set of nodes that are reachable from each other. The average neighborhood size is 3 nodes. We define the steady state of the algorithm as the state when the probabilities of the tests are stable. A test is stable if the probability of the test from the previous stage and the probability at the current stage are within epsilon of each other. In other words $P_k(T_{ij}) - P_{k-1}(T_{ij}) \leq \epsilon$. We computed the probability of selection for the nodes and compared it to the node vulnerability. The results are depicted in figures 13,14,15. For each node the plot of the probability of selection and the vulnerability is made. The most vulnerable node is node 4, and it has the highest probability of selection

among the 10 nodes in the network. Figure 13 shows the plots for node 4 and its neighbors. For node 4 the ratio of the probability of selection and the vulnerability converges to 1.00 at the steady state. For the two most vulnerable neighbors selected, namely, nodes 6 and 2, the ratio converges to 2.00. This results from the fact that whenever node 4 is selected, these two most vulnerable nodes are also selected, so their probability of selection is related to the probability of selection of node 4. As for node 3 the non-vulnerable neighbor, the ratio converges to 1.00 at stability.

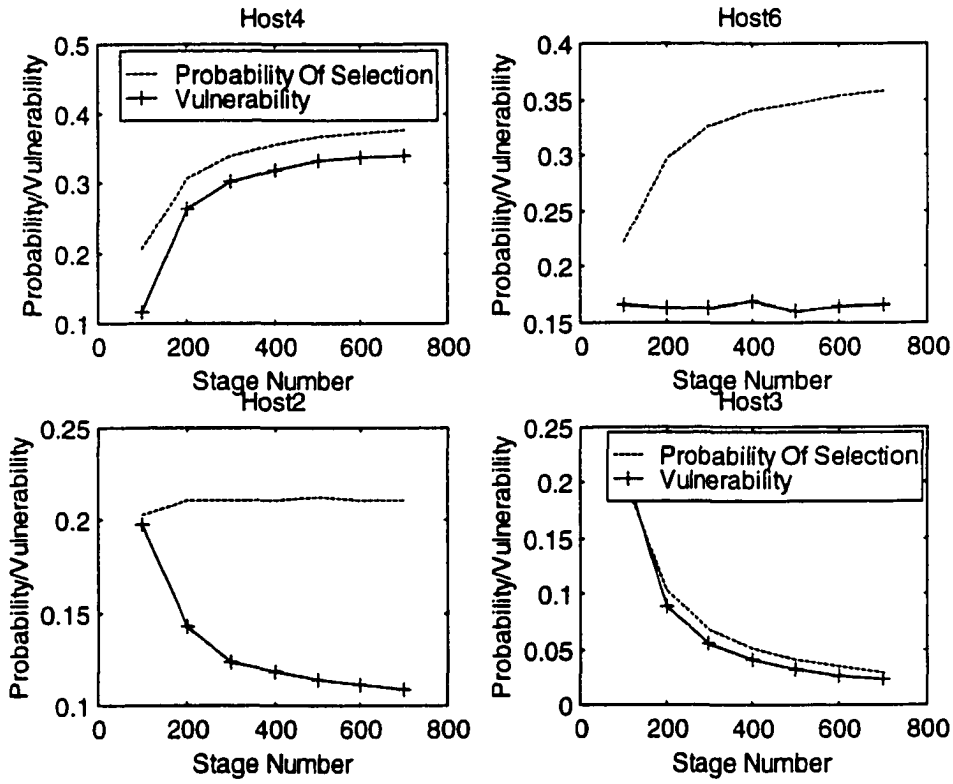

Figure 13. Probability of selection versus vulnerability

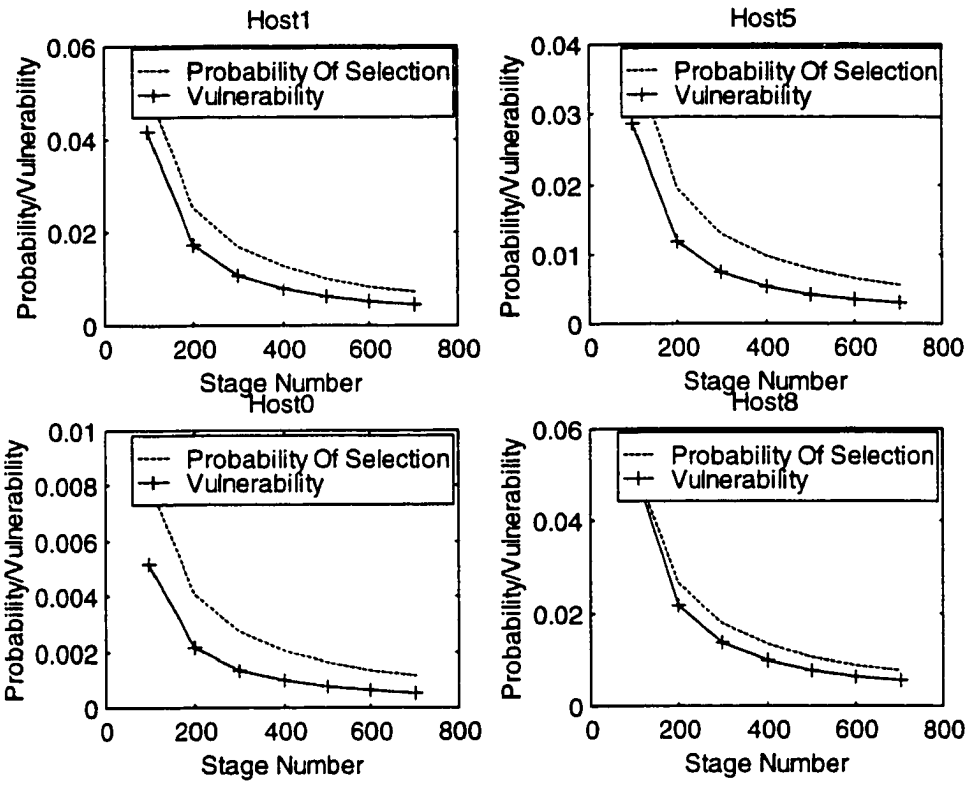

Figure 14. Probability of selection versus vulnerability

Figure 14 shows the plots of the probability of selection and vulnerability for nodes 1,5,0,8. Node 8 is one of the vulnerable nodes in the network, and the ratio of the probability of selection to the vulnerability for this node also converges to 1.00 at the steady state. As for the two most vulnerable neighbors of node 8,namely, nodes 1 and 5, the ratio converges to 1.8 at the steady state. Again this results from the fact that the selection of nodes 1 and 5 is related to the selection of node 8. Every time node 8

is selected these two nodes are also selected, since they are the two most vulnerable neighbors of node 8.

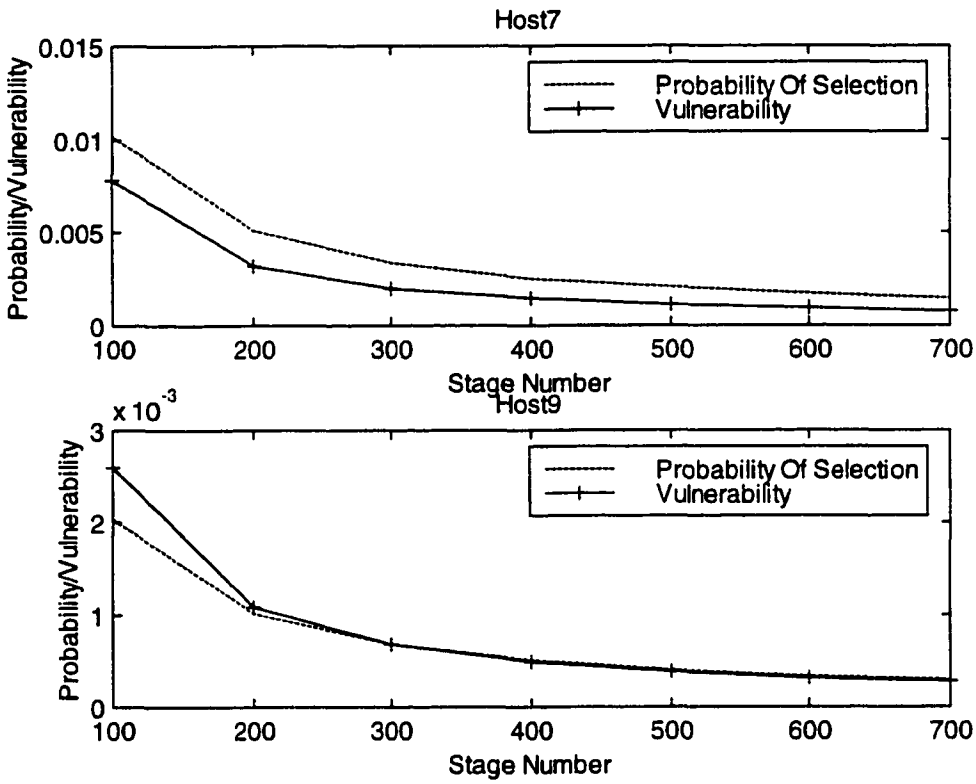

Figure 15. Probability of selection versus vulnerability

Finally figure 15. shows the plot of the probability of selection and vulnerability for nodes 7 and 9. These two nodes are non vulnerable nodes within the network. The ratio of the probability of selection to the vulnerability converges to 1.8 for node 7 and 1.1 for node 9 at steady state.

The probability of miss for the advanced two dimensional case is defined by the following formula:

$$1-\frac{1}{q}\sum_{k=1}^{q}\{[P_k(host(j)\mid P(T_{ij})\ is\ maximum\ \forall i=0,...,No\_tests,\forall j=0,...,h)]$$

$$+\sum[P_k(host(w)\mid P(T_{iw})is\ \max imum, w\in vul(j)\subset neighborhood(j))\}$$

where q is the number of stages executed, h is the number of nodes to be tested, max_h is the number of nodes selected for testing at each stage, $P_k(host(j))$ is the probability of selection for node j at stage k, $P(T_{ij})$ is the probability of test i being positive at node j, No_tests is the total number of tests to be performed, and neighborhood(j) is the set of nodes in the neighborhood of node j as defined above. vul(j) is a set of size G of the most vulnerable nodes in the neighborhood of node j. Thus for every vulnerable node j selected, the G most vulnerable neighbors are also selected.

Table 2 shows the probability of miss for different values of max_h.

Max_h	Prob(miss)
4	0.112
6	0.09
8	0.0

Table 2. Probability of miss

Increasing the number of nodes tested simultaneously (max_h), decreases the probability of miss. In fact from the results in table 2, only testing 6 nodes simultaneously gives a probability of miss of 0.09, which is quite good compared to the method of random selection were the probability of miss is 0.4.

5.3 Comparison of the One Dimensional and Advanced Two Dimensional cases.

We compare the one dimensional and the advanced two dimensional cases with respect to the probability of selection and the probability of miss. Figure 16 shows the plot of the probability of selection for the two most vulnerable nodes in the network, namely, nodes 4 and 6.

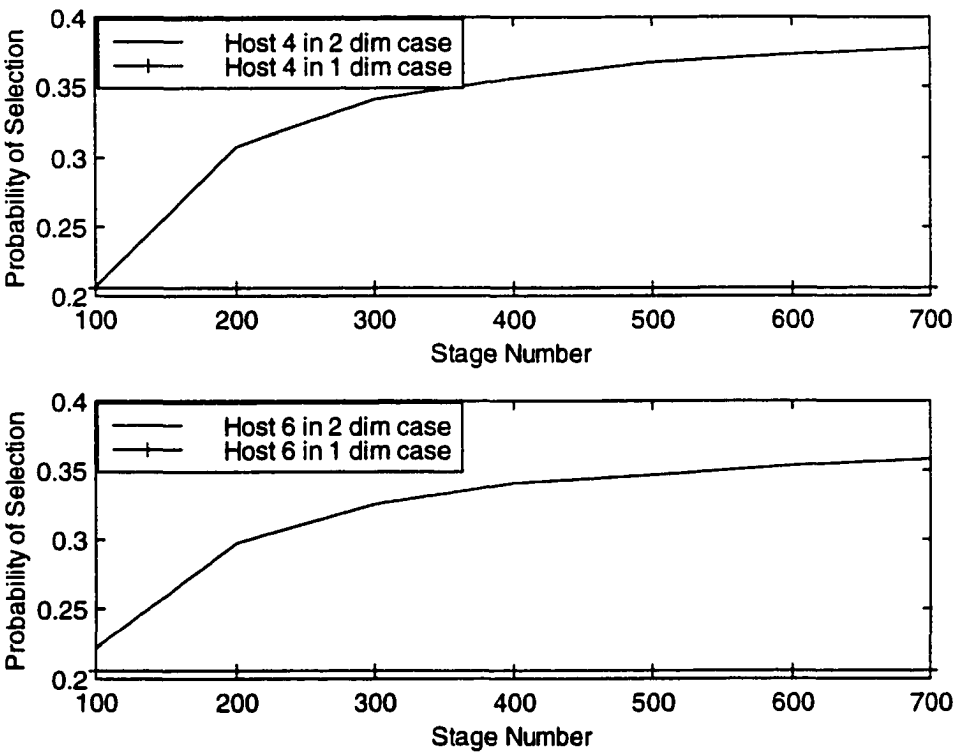

Figure 16. Probability of Selection

The advanced two dimensional case reaches the stable state after 700 stages of execution, compared to 1900 stages for the one dimensional case. For the most vulnerable node, node 4, the probability of selection in the two dimensional case is

1.84 times larger than the probability of selection for the same node in the one dimensional case. For node 6 the probability of selection in the two dimensional case is 1.7 times larger than in the one dimensional case. Thus the two dimensional case increases the probability of selection of the vulnerable nodes in the network. This in turn results in a lower probability of miss. This comes from the fact that for every vulnerable node selected, the G most vulnerable neighbors of that node are also selected for testing. This is apparent in the formula for the probability of miss.

Probability of miss for the two dimensional case:

$$1-\frac{1}{q}\sum_{k=1}^{q}\{[P_k(host(j)\mid P(T_{ij})\ is\ maximum\ \forall i=0,\ldots,No\_tests,\forall j=0,\ldots,h)]$$

$$+\sum[P_k(host(w)\mid P(T_{iw})is\ \max imum, w\in vul(j)\subset neighborhood(j))\}$$

Probability of miss for the one dimensional case:

$$1-\frac{1}{q}\sum_{k=1}^{q}P_k(host(j)\mid Pos(j)\ is\ maximum\ \forall i=0,\ldots,No\_tests,\forall j=0,\ldots,h)$$

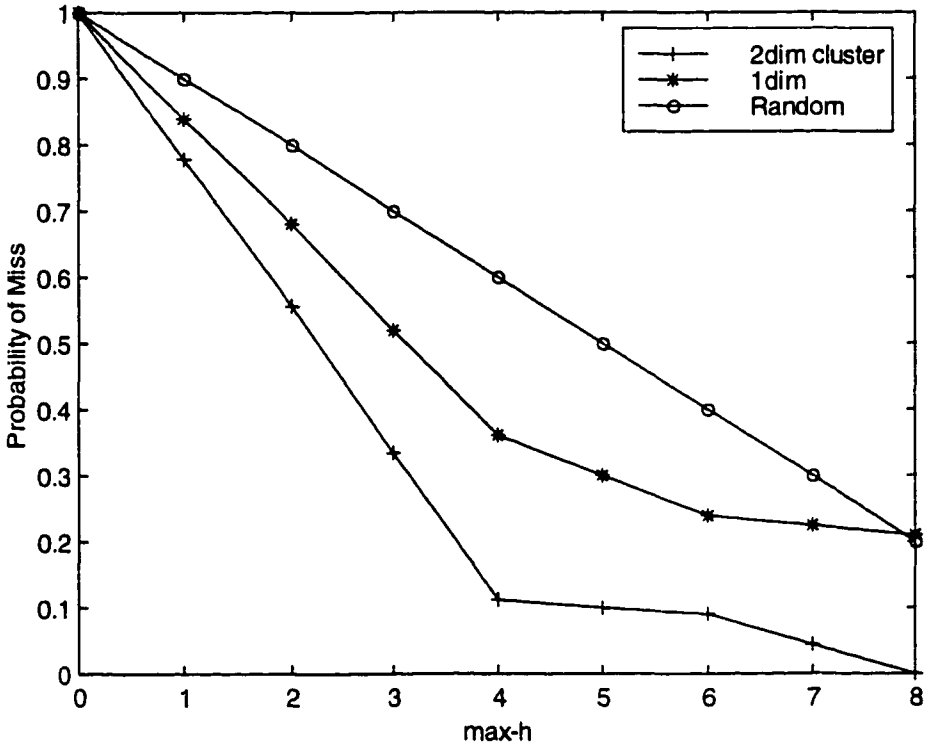

Figure 17. Comparison of Probability of Miss

The probability of miss for the one dimensional, advanced two dimensional and random cases are shown in figure 17. In the random case nodes are selected at random with no regard to their vulnerability. These values are computed for h=10 nodes. In all three cases increasing the number of nodes tested simultaneously, namely, max-h, decreases the probability of miss. Both the one and two dimensional cases give better results than the random case of node selection. However, the two dimensional case gives better results for the probability of miss than the one dimensional case. In choosing between testing schemes, the scheme that provides the

smallest probability of miss should be chosen. In section 5.1 we stated that in the case of a random selection scheme, where no consideration is taken into the vulnerability of the nodes in the selection process, the best value for the probability of miss we can expect is $1-\frac{max\_h}{h}$, this provides us with an upper bound for the probability of miss for our testing scheme.

The tradeoff between the one dimensional and advanced two dimensional schemes, is that although in the two dimensional scheme we are increasing the number of nodes tested simultaneously, we are actually saving both bandwidth and computing resources since one agent is assigned for testing a cluster of nodes within a neighborhood as opposed to dispatching 1+G agents; one agent for testing the selected vulnerable node, and the G agents for testing the G most vulnerable neighbors of the selected node. Increasing the number of nodes tested simultaneously also decreases the probability of miss compared to the one dimensional scheme.

In a brute force testing scheme where all the tests are applied to all the nodes simultaneously, the number of tests applied per stage would be No_tests*h, which for our example is 300 tests per stage. Because of the bandwidth and resource limitations imposed, this scheme is not feasible. In our advanced two dimensional scheme, the number of tests executed during each stage is $\frac{max\_h}{2}(2+G)$, where max_h is the number of nodes to be tested simultaneously, G is the number of vulnerable neighbors selected for each vulnerable node. Here we are assuming that $\frac{max\_h}{2}$ vulnerable

nodes are selected, and the remaining $\frac{\max\_h}{2}$ are selected from the negative, and new nodes. If the number of vulnerable nodes in the network is V, V<=h, then the advanced two dimensional scheme would select the $\frac{\max\_h}{2}$ most vulnerable nodes of these V nodes together with their most vulnerable neighbors. For a 10 node network (h=10) with G=2, and max_h=8, the number of nodes/tests applied per stage would be 16 in the two dimensional scheme, compared to 300 in the brute force scheme, and assuming that the number of vulnerable nodes is V=4, then the two dimensional scheme selects these four nodes in addition to their most vulnerable neighbors, which is indicated by the fact that the probability of miss for this case is 0. Thus the two dimensional scheme is able to test these vulnerable nodes using only 5% of the available test pool.

6. Conclusion

We have presented an intelligent mobile agent system for testing telecommunications networks. The system is a general purpose method that can be used for any type of network testing, for example, vulnerability assessment, intrusion detection. The system consists of mobile agents equipped with tests to be performed on targets in a network. The tests and the targets are selected using probabilistic reasoning in a manner that maximizes the probability of selection. The algorithm takes into consideration the fact that problems such as vulnerabilities and intrusions for example, tend to spread once they have entered the network. Thus once the algorithm detects a problematic node, it selects the most vulnerable nodes within the neighborhood of the selected node and applies the appropriate set of tests to them. This is crucial in telecommunication networks were bandwidth and resource limitations prohibit the application of large pools of tests simultaneously. Thus our method selects tests and targets in an optimum manner that ensures detection of any problems within the network in a timely and efficient manner, without overloading the network.

The probability of miss (not selecting) for the hosts was used as a measure of the algorithms' performance. Other measures were also used to study the performance of the algorithm; the ratio of the probability of host selection to the vulnerability of the host, another measure of performance was the ratio of the probability of selection for the tests to the probability of a test being positive

Results show that the method does optimize the selection process for both the hosts and the tests within the given limitations of bandwidth and resources.

7. Future Work

The future direction for this work is an actual real system implementation that employs actual mobile agents on a real computer network system. We are also looking into the use of routing information in the process of creating and dispatching mobile agents in an effort to minimize the number of agents disptached.

8. References.

[1] Jacobs, S., Dumas, D., Booth, W., Little, M., "Security Architecture for Intelligent Agent Based Vulnerability Analysis", MILCOM 1999.

[2] Conner, M., Patel, C., Little, M., "Genetic Algorithm/Artificial Life Evolution of Security Vulnerability Agents", MILCOM 1999.

[3] Barret, M., Little, M., Poylisher, A., Gaughan, M., Tardiff, A., "Intelligent Agents for Vulnerability Assessment of Computer Networks",*Proceedings of the 2nd Annual FedLab Symposium-Advanced Telecommunications & Information Distribution, U.S. Army Research Labs*, 1998.

[4] Pearl, J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1986.

[5] Khreisat, Laila., Dalal, M., "Anytime Reasoning with Probabilistic Inequalities", *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence*, Nov., 1997.

[6] Nilsson, N. J., Probabilistic Logic, *Artificial Intelligence*, 28(1):71-87, 1986.

[7] Fagin, R., Halpern, J. Y., and Megiddo, N., "A Logic for Reasoning about Probabilities", *Information and Control*, 87:78-128, 1990.

[8] Mendelson, E., *Introduction to Mathematical Logic*, Van Nostrand, Princeton, N.J., 1964.

[9] Karmarkar, N., "A New Polynomial-time Algorithm for Linear Programming", *Combinatorica*, 4:302-311, 1984.

[10] Pham, V., Karmouch, A., "Mobile Software Agents: An Overview", *IEEE Communications Magazine*", 36(7), July 1998.

[11] Khreisat, L., Saadawi, T., Lee, M., "*Anytime Probabilistic Reasoning Mobile Agent System for Network Testing*", ATIRP 2000.

[12] Kumar, S., "Classification and Detection of Computer Intrusions", Ph.D. Thesis, Purdue University, August 1995.

[13] Rothermel, K., Popescu-Zeletin, Eds., "Mobile Agents", lecture Notes in Comp. Sci. Series, vol. 1219, Springer 1997.

[14] Crosbie, M., Spafford, G., *"Defending a Computer System using Autonomous Agents"*, Tech. Report No. 95-022, Dept. Computer Science, Purdue University, 1994.

[15] Crosbie, M., Spafford, G., *"Active Defense of a Computer System using Autonomous Agents"*, Tech. Report No. 95-008, Dept. Computer Science, Purdue University, 1995.

[16] *Mobile Agents White Paper,* General Magic,1997.

[17] Saadawi, T.,Ammar, M.,Hakeem, A., *Fundamentals of Telecommunication Networks,* John Wiley & Sons,1994.

[18] Schwartz, M., *Telecommunication Networks:Protocols, Modeling and Analysis*,Addison-Wesly, 1987.

[19] Khreisat, L., Saadawi, T., Myung, L., *"Adaptive Probabilistic Reasoning Mobile Agent System for Network Testing: Two dimensional case"* submitted to IEEE JSAC 2000.