Evolutionary Computation Techniques for Intrusion Detection in Mobile Ad Hoc Networks

Sevil Sen[1], John A. Clark,

Abstract

Intrusion detection on mobile ad hoc networks (MANETs) is difficult. This is because of their dynamic nature, the lack of central points, and their highly resource-constrained nodes. In this paper we explore the use of evolutionary computation techniques, particularly genetic programming and grammatical evolution, to evolve intrusion detection programs for such challenging environments. Cognizant of the particular importance of power efficiency we analyze the power consumption of evolved programs and employ a multi-objective evolutionary algorithm to discover optimal trade-offs between intrusion detection ability and power consumption.

Keywords: Mobile ad hoc networks, intrusion detection, evolutionary computation, multi-objective optimization, power-aware detection, cooperative security

1. Introduction

Mobile ad hoc networks (MANETs) are one of the fastest growing areas of research. This new type of self-organizing network combines wireless communication with a high degree of node mobility. Unlike conventional networks, they do not have fixed infrastructure elements such as base stations or centralized management points. Rather, they rely on dynamically determined collaborations to provide communications between nodes and so provide access to services distributed across the network. Such a modus operandi makes MANETs attractive for many applications. For example, MANETs might find natural deployment where the existing/fixed infrastructure can be non-operational due to natural disasters, war, etc. Rapid service provision to conference attendees is also possible; setting up a network infrastructure would otherwise take a great deal of time. One area that is proving particularly challenging for MANETs is intrusion detection. In this paper, we investigate the use of evolutionary computation techniques to synthesise intrusion detection programs suited for use with MANETs. Of particular note is our goal of taking into account not only detection capabilities but also resource consumption. Thus, power usage is a particularly important concern. (This issue is exacerbated when future networks comprising smart dust are concerned.)

We show that evolutionary computation techniques (primarily Genetic Programming and Grammatical Evolution) can be effective means to obtain intrusion detectors. Furthermore, we demonstrate how these and other evolutionary techniques can be used to explore the trade-offs between competing performance objectives, e.g. how power usage can be traded against detection performance. We know of no other means by which this sort of exploration can be reliably achieved. As proof of concept we concentrate on the detection of two attacks on routing protocols in MANETs.

1.1. Attacks on MANETs

MANETs share the vulnerabilities of wired networks, such as eavesdropping, denial of service, spoofing and the like; these are simply accentuated by the ad hoc context [1]. In this paper we focus on attacks which are more specific to MANET operation, namely attacks on MANET routing protocols. Conventional networks use dedicated nodes to carry out basic functions like packet forwarding, routing, and network management. In ad hoc networks, however, these are carried out by all available nodes. Mobile nodes that are within each other's radio range can communicate directly via wireless links, while those that are far apart must rely on other nodes to relay messages. (This is usually referred to as multi-hop communication.) The mobility of network nodes forces routes between nodes to be updated frequently and various protocols have been designed for finding/updating routes and providing communication between endpoints through cooperating intermediate nodes.

Routing protocols on MANETs are of two basic forms: proactive (*e.g.* OLSR [2]) and reactive protocols (*e.g.* AODV [3], DSR [4]). Hybrid approaches are also possible. Attacks on these protocols can be implemented easily; they do not require physical access to a wired medium since all communication is wireless, and so form an important threat consideration. In addition, they pose specific technical challenges. For example, the dynamic topology of MANETs makes it harder to differentiate normal behaviour of the network from anomalous behaviour; packets may be lost due to malicious intervention, or simply because a route is no longer feasible. Furthermore, routing algorithms for MANETs usually assume that nodes are cooperative and non-malicious. This presents significant security problems, for example, a malicious node can easily become an important

Email addresses: ssen@cs.hacettepe.edu.tr (Sevil Sen), jac@cs.york.ac.uk (John A. Clark)

routing agent and disrupt network operations by disobeying the protocol specifications.

1.2. Intrusion Detection

Intrusion is any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource [5] and an intrusion detection system (IDS) is a system for the detection of such intrusions. An IDS detects possible violations of a security policy by monitoring system activities. If an attack can be detected a response can be initiated to prevent or minimize the damage to the system. [1]

There are three main intrusion detection approaches: anomaly-based; misuse-based; and specification-based. An anomaly-based technique profiles the symptoms of normal behaviours of the system such as command invocation frequencies and CPU usage for programs. It detects intrusions as anomalies, *i.e.* deviations from established normal behaviours. A variety of techniques have been used to implement anomaly detection, e.g. statistical approaches, and artificial intelligence techniques like data mining and neural networks. Misuse-based detection compares known attack signatures with current system activities. It is generally preferred by commercial IDSs since it is efficient and has a low false positive rate. With specification-based approaches a set of constraints of a program or a protocol are specified and intrusions are detected as runtime violations of these specifications. They combine the strengths of anomaly-based and misuse-based detection techniques [6], providing detection of known and unknown attacks with lower false positive rate.

2. Related Work

Applications of evolutionary computation techniques to intrusion detection on conventional networks have usually employed either genetic programming (GP) or genetic algorithms (GA). The first GP application to intrusion detection is given by Crosbie and Stafford [7]. The main idea in that research is to train autonomous agents based on the input features and the functions given to detect intrusive behaviours. Obvious intrusions that are mis-classified during the evolution process are penalised heavily. There are also promising applications of genetic algorithms proposed for misuse-based intrusion detection systems [8][9].

Two recent approaches use GP and evaluate the evolved programs on the KDD-99 data set [10], which is the most widely used benchmark evaluation data for intrusion detection. In [11] the output program evolved by GP is small, simple and uses just a few input features where "most machine learning paradigms (artificial neural networks, support vector machines, decision trees) examine all input features to detect intrusions" [11]. Evaluation shows that the approach is lightweight and effective, satisfying the main goals of an intrusion detection algorithm. The GP techniques used in that research are compared with other machine learning techniques (Support Vector Machines and Decision Trees) for intrusion detection in [12]. The results show that genetic programming techniques outperform these techniques and are lightweight. In [13] linear GP is efficiently trained on a large data set by using RSS-DSS (Random Subset Selection-Dynamic Subset Selection) algorithm. This approach also uses a small set of the features.

Grammatical evolution (GE), another evolutionary computation paradigm, has been proposed recently for intrusion detection on wired networks [14]. It has been applied to the KDD-99 data set [10] and evolves detectors for different classes of attacks such as denial of service (DoS) and probe attacks. While the classification accuracy is higher for the DoS attack class, the U2R (user-to-root) and R2L (remote-to-local) attacks show low detection rates. The application of GE to intrusion detection is in its early stages and improvements are very likely possible.

Many IDSs have been proposed for wired networks. However, the specific features of MANETs make direct application of these approaches to our target networks problematic. Consequently, over the last decade researchers have proposed new approaches for intrusion detection in MANETs. One of the most commonly proposed techniques on MANETs is specification-based intrusion detection, where intrusions are detected as runtime violations of the specifications of routing protocols. This technique has been applied to a variety of routing protocols on MANETs such as AODV and OLSR [15][16]. A few signature-based IDSs have also been proposed. In [17], an approach is proposed based on a stateful misuse detection technique and which defines state transition machines for detecting known attacks on AODV. In [17], an IDS is proposed which uses a specification-based technique for attacks that violate the specifications of AODV directly and an anomaly-based technique for other kinds of attacks such as DoS. Since wireless nodes can overhear traffic in their communication range, promiscuous monitoring can also be used to detect MANET attacks such as dropping and modification [18][19][20][21]. Mobile agents have been suggested as another way to provide communication between IDS agents [22].

Few artificial intelligence based intrusion detection systems have been proposed to explore the complex behvioural space that MANETs provide. In the first proposed IDS for MANETs [23], statistical anomaly-based detection is chosen over misuse-based detection, since expert rules can detect only known attacks and the rules cannot easily be updated across a wireless ad hoc network. The SVM Light and RIPPER classifiers are employed and compared in that research. In [24] a Markov-chain based local anomaly detection model is proposed for a Zone-Based IDS architecture. The network is partitioned into zones based on geographic location. Another approach which constructs an anomaly-detection model automatically by extracting the correlations among monitored features is proposed in [25]. Furthermore, they introduce simple rules to determine attack types and sometimes attackers after detecting an attack using cross-feature analysis. A fuller account of previous work on intrusion detection in MANETs can be found in [26].

[1] It is fair to say that the issue of detection has been researched to a far greater degree than that of how to respond to detected attacks. As new types of system emerge, such as various autonomous systems with a requirement for continued operation, this research gap may become acute. However, the research in this paper is concerned only with detection.

Evolutionary computation techniques have been seen some application to problems in sensor networks but have not been applied to intrusion detection problem. In [27], the Distributed Genetic Programming Framework (DGPF) which automatically discovers distributed algorithms for given problems is introduced for sensor networks. The election problem is solved by using this framework as an example and a multi-objective optimization technique is also employed on this problem by considering non-functional fitness functions such as code and memory size. Further research develops a parallel evolutionary algorithm for suitable sensor networks in [28]. Applications of grammatical evolution and genetic programming on intrusion detection for MANETs have been proposed in our previous work [29][30]. This paper extends these applications and compares the performance of these techniques on intrusion detection. Furthermore it investigates a suitable intrusion detection architecture for MANETs.

2.1. Our Work: Ackowledging the Effects of Complex Environments

How to detect intrusions effectively and efficiently on MANETs is a challenging research problem. The dynamic nature of MANETs, the lack of central points and its highly resource-constrained environment are the main issues. Researchers have generally focused on the first two issues so far. However consideration of resource-constraints is vital. In this research the limited resources of nodes are also taken into consideration.

This research shows that GP and GE can be used to evolve efficient detectors for known attacks (ad hoc flooding and route disruption) against routing protocol on MANETs. The results are evaluated under different types of networks and compared with hand-coded programs. To the best of our knowledge it is the first application of evolutionary computation techniques to intrusion detection in MANETs.

This research proposes a novel approach by discovering different trade-offs between functional and non-functional properties of detection programs. Our main contribution in this research is to evolve a set of programs for each attack offering different trade-offs between intrusion detection ability and energy usage, since power is one of the critical resources. Moreover, we investigate if it is better to evolve separate programs for each attack or one program to detect both attacks. Our techniques can be used to generate solution sets with the best (or near best) trade-offs possible.

A suitable intrusion detection architecture is also investigated. Even though the same architecture has been investigated before in other approaches, it is a novel approach in terms of proposing how to choose monitoring nodes in MANETs by considering limited resources. Besides energy consumption of programs, energy consumption by message sending and receiving in cooperative detection is taken into account. The interaction between IDS agents and the number of nodes participating in detection is reduced by using multi-objective optimization techniques.

To conclude, we demonstrate the potential use of evolutionary computation techniques to discover complex properties of MANETs and to propose a suitable intrusion detection on this new environment.

3. AODV and Target Attacks

The Ad-hoc On-demand Distance Vector (AODV) routing protocol [3] is one of the most commonly used on-demand routing protocols on MANETs. It is used as an exemplar routing protocol in our research. We now briefly outline the operations of AODV and describe the two attacks on it that are the focus of our research. A fuller account of threats to MANETs can be found in [31].

Ad Hoc Flooding Attack. Network topology changes frequently on MANETs due to mobility. Link breakages may make existing routes inactive and initiate discovery of new routes by route request packets (RREQ). RREQ messages are sent only when nodes need a new route in reactive routing protocols such as AODV. Evidently, mobility may increase the number of RREQ packets on the network. In the flooding attack scenario, the attacker exploits this property of the route discovery mechanism by broadcasting a lot of RREQ messages for randomly selected nodes. The attacker aims to consume the resources of the nodes and the network. We believe that high mobility makes it difficult to distinguish flooding attacks from benign behaviour on a network, since it may also cause a high number of RREQ packets in the network. In our simulations, the attacker broadcasts 20 route request packets in a row as in [32].

Route Disruption Attack. In this attack, the attacker sends route reply (RREP) messages to the victim node without receiving any route request messages from that node. Instead of sending route replies for random destination nodes, the attacker chooses one of its neighbours as a victim and sends fresher RREP messages (with higher destination sequence numbers) to this node for disrupting the active routes in its routing table. Since the attacker is the victim node's neighbour, he already knows about the active routes of the victim through the routing control packets broadcast by that node. As stated in [24], one or few routing control packets could hardly incur severe damage to the system. So, in the simulation the attacker sends 5-10 RREP packets to the victim in a time interval.

In this research the networks are simulated by *ns-2* [33]. Mobility patterns of the nodes are simulated by the Random Waypoint model which is created using *BonnMotion* [34]. In the Random Waypoint model, each node moves from its current location to a random new location with random speed and pause time in determined speed/pause time limits [35]. Different network scenarios are created with different mobility levels and traffic loads. 50 nodes are placed in a topology of 1000m by 500m. TCP traffic is used for communication. The maximum number of connections is set to 20, 30 and 40 to simulate different traffic loads. The maximum speed of nodes is set to 20 m/sec and the pause time between movements is set to 40, 20, and 5 sec to simulate low, medium, and high mobility respectively. AODV periodic hello messages are used for local link connectivity. The simulations run 5000 seconds for training and 2000 seconds for testing.

Evolutionary computation techniques will be employed to create artifacts that detect these two attacks effectively and efficiently. These techniques are described in the next section.

4. Introduction to Evolutionary Computation

Evolutionary Computation (EC) is a research area inspired by natural evolution. It is loosely based on the process of Darwinian *survival of the fittest*, where individuals are competing with each other for survival and reproduction in an environment that can only host limited number of individuals [36]. Evolutionary computation uses this approach to create solutions for a given problem automatically where candidate solutions of the problem correspond to the individuals, and the best solutions correspond to the fittest individuals in a population. The general steps in evolutionary computation are summarised below.

initialize population
while termination criterion not satisfied **do**
  execute and evaluate fitness value of each individual
  apply genetic operators (selection, crossover, reproduction, mutation, etc.) to the individuals
  create new population
end while

EC starts with generating a population of individuals (usually randomly) which are candidate solutions for the target problem. Then, each individual is evaluated and assigned a *fitness value* that indicates how well this candidate solves or comes close to solving the problem at hand. Until a termination criterion is satisfied, new populations are generated iteratively by using selection, crossover, and mutation operators as in the natural evolution. These genetic operators are used to provide better solutions in the new population. Selection provides a great opportunity for fitter individuals to survive by picking out individuals based on the fitness value from the current population for mating (with fitter candidates having greater chances of selection). Whilst crossover mimics the exchange of DNA under sexual production to generate new individuals, mutation mimics natural mutation by changing selected individuals to introduce diversity into the population.

Some key parameters need to be defined to solve a problem by using evolutionary computation techniques. *Population size* is the number of individuals in a population in one generation. *Generations* defines when (at which generation) the evolution process terminates. Terminating does not necessarily happen at a defined generation. There can be other terminating conditions, such as termination upon finding an ideal candidate (i.e. a true solution). Since finding the ideal solution can take a very long time for complex problems, the parameter *generations* is typically used as in our research. There are also other parameters which define genetic operators. The *crossover probability* shows how likely this operator will be performed on the individuals selected for mating (and thus provide a direct means of allowing current candidates to proceed to the next generation prior to any mutation). The *mutation probability* shows how likely a part of an individual will be mutated. The *reproduction probability* shows how likely an individual will be copied without any modification to the new generation. For the selection operation, there are different methods (e.g. roulette-wheel, rank-based, tournament selection) and different parameters for each method respectively. Tournament selection is used in our experiments. In this method, a group of individuals is chosen randomly from the population and the fittest individual from this group is selected as a parent. *Tournament size* defines the number of the individuals in this group.

In EC the population size is typically constant most of the time, so the individuals who will survive in the next generation need to be selected. A choice is made among the current population and the new individuals generated by variation operators. This choice is based on the fitness value. There are two main replacement mechanisms: simple, and steady-state. In a *simple* replacement approach, the new individuals (children) replace the current population. In the *steady-state approach*, only one individual, which is generally the worst member of the population, is replaced. Hence, in the latter method, the best fitness value of the population steadily increases (or stays still) as the number of generations increases.

4.1. Genetic Programming

Genetic programming (GP), popularised significantly by Koza [37], is one of the most widely employed evolutionary computation techniques in the literature. It is claimed that GP has equaled or exceeded the performance of other machine learning techniques, and also evolved better programs than the best programs written by people [38]. It has been applied to many problems.[2]

The individuals are typically represented by a tree structure in GP. A GP tree is built from functions and terminals. Terminals are the leaves in a tree and are generally the inputs to the GP, constants, or other functions with no argument. Functions can be mathematical operators, boolean functions, program statements (if, loop), and the like. *Tree depth* defines the maximum size of the individuals (trees) and is the length of the longest path in the tree from the root node.

4.2. Grammatical Evolution

Grammatical Evolution (GE) is a technique that allows us to generate programs in an arbitrary language by evolving programs written in a BNF grammar [39]. GE is not the first technique using grammars, but it presents a unique way of using grammars in an evolution process [40]. The core idea of GE relates to how simple integer sequences can be interpreted as programs.

[2]The interested reader may consult the Human Competitive Awards reference page for details of particularly successful applications of GP and other evolutionary techniques.

4.3. Multi-Objective Evolutionary Computation

Multi-Objective Optimisation aims to optimise two or more, often conflicting objectives simultaneously. The solution to a multi-objective optimisation problem is often not unique. It is the set of optimal solutions called the Pareto set. Here an objective vector x is said to dominate another objective vector y ($x \succ y$) if each criterion (objective value) of x is no greater than the corresponding component of y and at least one criterion is less (lesser values are preferable).

$$x \succ y : if\ x_i \leq y_i\ for\ each\ i\ and\ x_i < y_i\ for\ some\ i \quad (1)$$

The Pareto front comprises the solutions that are not dominated by any other individuals. In other words, it includes the optimal solutions (non-dominated) which represent different trade-offs among the objectives.

Multi-Objective Evolutionary Computation (MOEC) allows us to combine multi-objective optimisation with evolutionary search. SPEA2 [41] is one of the most popular Pareto-based MOEC algorithms. An implementation of SPEA2 which is an extension to ECJ [42] is utilized in our research.

4.4. Why Evolutionary Computation?

MANETs are a new type of distributed network whose properties are complex. It is hard to distinguish attacks from normal activities under such dynamic environments. It is far from clear whether the human perception of what makes a good intrusion detection algorithm in these contexts really is the best possible. Moreover resource-constrained nodes require different trade-offs to be made between intrusion detection ability of programs and their resource usage. Humans are not particularly adept at selecting good choices when complex trade-offs have to be made.

In this research evolutionary computation techniques are proposed to discover automatically complex properties of MANETs. Although various artificial intelligence technqiues have been proposed for intrusion detection, EC is one of the most promising approaches. It makes fewer assumptions about the solution space. IDS programs derived using GP or GE lend themselves to some degree to manual analysis. We can often see what the program is doing. Furthermore recent research shows that the programs evolved by EC are lightweight and use far fewer features compared to some other machine learning techniques. These characteristics are among the main motivations behind using EC in this research. Futhermore multi-objective evolutionary algorithms allow us to optimize multiple objectives simultaneously, so they can be used to discover detection programs that are both effective (*i.e.* detect intrusions without a high false positive rate) but also efficient (in particular, suited to deployment on constrained resource platforms). These features make EC very attractive for the development of intrusion detection programs suitable for MANETs.

5. Evolving Intrusion Detection Rules

In this research, we mainly aim to investigate the use of evolutionary computation techniques to evolve intrusion detection rules for known attacks against MANETs. Genetic programming and grammatical evolution are employed to evolve programs that detect the ad hoc flooding and route disruption attacks described above. These techniques are evaluated on simulated networks with varying traffic and mobility patterns, and compared. The application of each technique to intrusion detection in MANETs is detailed in the subsequent sections.

5.1. Feature Selection

"Features" are characteristics of our system whose measurements provide the inputs to our evolved decision algorithms. They provide the basic data any such evolved algorithms can use to reach a result. The choice of which characteristics can be used for these purposes is very important. They must contain sufficient information to allow the fundamentals to be developed.

Table 1: The Features

Features	Explanation
neighbours	no. of neighbours
added_neighbours	no. of added neighbours
removed_neighbours	no. of removed neighbours
active_routes	no. of active routes
repaired_routes	no. of routes under repair
invalidated_routes	no. of invalidated routes
addedroutes_disc	no. of added routes by route discovery mechanism
addedroutes_notice	no. of added routes by overhearing
updated_routes	no. of updated routes (modifying hop count, sequence number)
added_repairedroutes	no. of added routes under repair
invroutes_timeout	no. of invalidated routes due to expiry
invroutes_other	no. of invalidated routes due to other reasons
avg_hopcount	average no. of hop counts of active routes
recv_rreqPs	no. of received route request packets destined to this node
recvF_rreqPs	no. of received route request packets to be forwarded by this node
send_rreqPs	no. of broadcasted route request packets from this node
frw_rreqPs	no. of forwarded route request packets from this node
recv_rrepPs	no. of received route reply packets destined to this node
recvF_rrepPs	no. of received route reply packets to be forwarded by this node
send_rrepPs	no. of initiated route reply packets from this node
frw_rrepPs	no. of forwarded route reply packets from this node
recvB_rerrPs	no. of received broadcast route error packets (to be forwarded or not)
send_rerrPs	no. of broadcasted route error packets from this node
recv_aodvPs	no. of received total routing protocol packets
recvF_aodvPs	no. of received total routing protocol packets to be forwarded
send_aodvPs	no. of initiated total routing protocol packets from this node
frw_aodvPs	no. of forwarded total routing protocol packets by this node

We have opted for a considerable degree of expressiveness. Table 1 shows the features maintained at each node by the routing protocol. (Some additional information is also stored, *e.g.* some configuration constants, but these are of little use for our purposes and so are omitted from the table.) We provide a rich set of features and expect our techniques to select judiciously from them. It would be difficult to confidently supply apriori a narrower set of features. We allow the evolutionary computation algorithms to choose necessary features among those available.

The features can be categorized into two main groups: mobility-related and packet-related features. Mobility-related features help reflect the mobility model of a node or the network. Some of the mobility features give information about mobility directly such as changes in the number of neighbours. Others can be the results of mobility such as changes in the

routing table (*e.g.* number of new routes, number of invalidated routes) in a time interval. Packet-related features include information about the frequency of the routing protocol control packets (RREQ, RREP, RERR) sent, received, or forwarded in a time interval. The average hop count feature is used only for detection of route disruption attacks. These features are gathered periodically by each node. All features are local to a node, so no communication with other nodes is needed to gather them.

5.2. Application of GP to Intrusion Detection in MANETs

In GP a problem is defined with functions and terminals (features) which are the parts of a GP tree, and the fitness function. Here we use strongly-typed GP (STGP) [43]. STGP enforces data type constraints. For example a generic function could be forced to use specific data types (as input and output). The feature set given in Table 1 is used as the terminal set. The operators applied to these features are the basic mathematical (add, sub, mult, div, pow, min, max, percent, sin, cos, log, ln, sqrt, abs, exp, ceil,floor), relational, and comparison operators.

In our experiments we use a fitness function based on the main metrics used to evaluate an IDS (*i.e.* detection rate, false positive rate) as shown in the equation below. The detection rate (DR) shows the ratio of correctly detected intrusions to the total intrusions on the network. The false positive rate (FPR) shows the ratio of normal activities that are incorrectly marked as intrusions to the total normal activities on the network. An acceptable low rate of false alarms is as important as a high detection rate. A high false positive rate will cause a good deal of time to be wasted and will likely destroy confidence in the IDS.

$$Fitness = \quad detection\ rate - false\ positive\ rate \qquad (2)$$

We assume that attacks are detected by the nodes that the attacks affect directly. In flooding attacks the nodes who are flooded by route request messages detect the attack. In route disruption attacks, the victim node is assumed to detect malicious change in its routing table.

We use the toolkit *ECJ 18* [42] for the GP implementation in our experiments. The GP parameters are selected as follows: 100 for population size, 1000 for generations, 0.9 for crossover probability, 0.1 for reproduction probability and 7 for tournament size. The parameters not mentioned here are the default parameters of the toolkit.

5.3. Application of GE to Intrusion Detection in MANETs

In GE, a problem is defined with a grammar and a fitness function. In this research the grammar in Table 2 is used to evolve programs in order to detect the identified attacks (ad hoc flooding, route disruption) on MANETs and raise an alarm.

This grammar defines an 'if' statement. The variables and the functions used in this grammar are the same as those used in GP. Even though more complicated grammars including 'if-else' statements, loops are employed at first, it is observed that simplified programs make the evolution process much easier. Moreover the grammar in Table 2 shows a good performance in detecting attacks as shown in the results section. The fitness function given in Equation 2 is used in the GE algorithm as well.

Table 2: BNF grammar used for the problem

```
S = <code>
<code> ::= if(<cond>) {raise_alarm()}
<cond> ::= <cond><set-op><cond> | <expr><relop><expr>
<expr> ::= <expr><op><expr> | (<expr> <op><expr>) |
           <pre-op>(<expr>) | <pre-op2>(<expr>) | <var>
<op>      ::= + | - | / | *
<pre-op> ::= sin | cos | log | ln | sqrt | abs | exp | ceil | floor
<pre-op2>::= max | min | pow | percent
<rel-op>  ::= < | ≤ | > | ≥ | == | !=
<set-op>  ::= and | or
<var>     ::= feature set in Table 1
```

We use the library *libge* [44] for the GE implementation in our experiments. The GE parameters used in GE are as follows: 100 for population size, 2000 for generations, 0.9 for crossover probability, and 0.01 for mutation probability. A steady-state approach is employed for replacement mechanism. Other parameters are the default parameters of the toolkit.

6. Performance Evaluation

6.1. Results Analysis

For each attack we evolve separate programs using GP and GE. For training, we use the network simulation under medium mobility with 30 TCP connections. The same network with attacks and without attacks is used together for training to reduce false positives. The best result of ten runs is chosen and distributed to each node on the network.

Table 3: Performance of GP and GE techniques on intrusion detection

Network	**Flooding Attack**				**Route Disruption Attack**			
	GP		**GE**		**GP**		**GE**	
	DR	**FPR**	**DR**	**FPR**	**DR**	**FPR**	**DR**	**FPR**
low mobility low traffic	99.81%	0.34%	99.81%	0.29%	100%	0.51%	100%	0.41%
low mobility med traffic	99.24%	1.94%	98.54%	1.72%	100%	0.99%	100%	0.88%
low mobility high traffic	98.60%	2.31%	98.14%	1.99%	100%	1.25%	100%	1.20%
med mobility low traffic	99.95%	0.36%	99.86%	0.36%	97.06%	0.46%	100%	0.40%
med mobility med traffic	99.89%	1.88%	99.86%	2.00%	100%	0.88%	100%	0.83%
med mobility high traffic	99.24%	2.97%	98.59%	2.79%	100%	1.27%	100%	1.18%
high mobility low traffic	99.79%	0.66%	99.96%	0.66%	100%	0.52%	100%	0.44%
high mobility med traffic	98.62%	1.83%	98.66%	1.73%	100%	0.84%	100%	0.76%
high mobility high traffic	99.74%	3.21%	99.65%	2.91%	100%	1.10%	100%	1.01%

Experimental results of the evolved programs (best individuals of each technique) are given in Table 3. Both techniques show a good attack detection performance. Route disruption

attacks seem to be easier to detect than ad hoc flooding attacks. In many cases the detection rate is 100% and the false positive rate is less than 1% (except under high traffic). For medium mobility and low traffic perfect detection (GP) is not reached, but the false positive rate is low (0.46%). It seems reasonable to suppose that a 100% detection rate can be achieved with a small increase in the false positive rate as observed in our further experiments. It is noted that route disruption attacks are quite evident by the nodes receiving forged route reply packets since they violate the specifications of the routing protocol directly.

For ad hoc flooding almost all cases give detection rates higher than a 99% while keeping the FPR reasonably low. Note that in both attacks the main difficulty seems to come from the traffic load: regardless of the mobility pattern, the FPR for medium/high traffic is always higher than for low traffic. This is a common characteristic of any detection technique which does not achieve a perfect detection, as the greater the traffic to be analysed, the higher the FPR.

The false positive rate changes due to the mobility and the traffic load. However they are not the only factors affecting it. Other factors such as network topology, traffic and mobility patterns also play a part. For instance, it is expected that the number of route request packets will be much higher in high mobility networks, since mobility could frequently make existing routes inactive. In Figure 1 the number of route request packets on a normal network and on a network under ad hoc flooding attack are presented. It shows little difference in the numbers of route request packets among the networks with different mobility levels. The network under low/medium mobility may also broadcast a lot of route request packets (sometimes higher than the network under high mobility) due to its topology, its mobility and traffic patterns to build and preserve its active routes. In conclusion, mobility is not the only major factor affecting the number of route request packets on the network. Hence the performance of evolved programs here is affected by those factors as well.

Benign nodes do not send more than four route request packets in a row in normal circumstances (in all cases of mobility). The important point here is how a node is affected by RREQ packets received under normal circumstances and under ad hoc flooding attack. Under high mobility nodes can receive many RREQs initiated from different nodes too. Under attack, the average number of RREQ packets received is increased up to 62.1% for low mobility, 61.8% for medium mobility and 41.5% for high mobility under low traffic. For medium traffic, it is increased up to 38.1% under low mobility, 42.8% under medium mobility and 35.7% under high mobility. For high traffic, it is increased up to 33.5% under low mobility, 26.7% under medium mobility and 21.9% under high mobility. As it is seen in the results, the difference get smaller for the networks under high mobility or high traffic networks.

Another factor affecting the performance of evolutionary computation techniques could be the fitness function. Since the number of normal events is much higher than the number of malicious events in the network, a small improvement in increasing the number of attacks detected could improve the fitness function much more than a small improvement in decreasing the number of false positives. The results also support this idea by evolving programs with high detection rates. This issue could be addressed by improving the fitness function such as assigning different weights to detection rate and false positive rate, or adapting a multi-objective fitness approach. In the sections 7-8 a multi-objective fitness function, which aims to optimize these multiple objectives simultaneously, is employed.

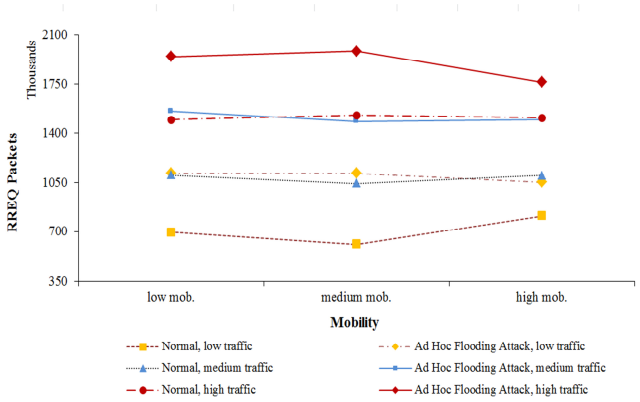

Figure 1: Route Request Packets on Simulated Networks

6.2. *The Performance of Manual Detection*

The performance of hand-coded programs for ad hoc flooding and route disruption attacks is evaluated in this section. In manual detection of an ad hoc flooding attack a threshold-based signature is typically used which simply considers the excessive amount of forwarded route request packets by a node (*frw_rreqPs* > *threshold*). Threshold-based signatures to detect resource depletion attacks in MANETs have already been employed in other approaches [17]. The performance of the signature is evaluated with different threshold values on a network with medium mobility and traffic. It is shown that the fitness value has its optimal value at the threshold value three, then it starts decreasing. The manual signature (using the threshold value three) is evaluated on networks with varying mobility and traffic patterns and demonstrated in Table 4. The results show that the manual detection achieves almost a perfect detection rate. However it does not perform well on differentiating benign flooding from malicious flooding, and results in non-negligible false positive rate for networks under high traffic. On the other hand GP and GE decrease the false positive rate with a small decrease in the detection rate. The false positive rate is decreased to almost half that of the manual detection. The difference becomes more remarkable with the distributed and cooperative intrusion detection programs evolved in Section 8.2. Overall, GP and GE outperform the manual detection for the ad hoc flooding attack by some considerable margin.

An intuitive signature which checks if the number of received route reply packets is consistent with (less than) the number of sent route request packets by a node is employed in the manual detection of route disruption attack (*init_rreqPs* < *recv_rrepPs*).

A similar approach has been used in some specification-based IDSs proposed for MANETs [15] which monitor request-reply flow of each routing packet. The results are demostrated in Table 4. The route disruption is a simple attack detected by small and relatively simple programs (such as those evolved earlier by GP and GE, and with results summarised in Table 3). The highest detection rate and a false positive rate less than 1% could be achieved by manual detection as well (except under high traffic/mobility). GE decreases the false positive rate a bit more (with reduction of up to 0.12%), while GP shows almost the same performance as manual detection. A node which compares the number of route reply packets sent and received for each node on the network separately can achieve a better detection performance, but this approach needs to hold a separate counter for each node in the network separately (while we only hold one counter here). The number of counter is increased proportional to the number of nodes in the network size. In this research the hand-coded signatures use only the features in Table 1 to allow a fair comparison. In conclusion GE shows a slightly better performance in detection of the route disruption attack than manual detection.

Table 4: Performance of Manual Detection on Simulated Networks

Network Scenarios	**Flooding Attack**		**Route Disruption Attack**	
	DR	FPR	DR	FPR
low mobility low traffic	99.95%	0.62%	100%	0.51%
low mobility med traffic	99.92%	3.19%	100%	1.00%
low mobility high traffic	98.94%	4.69%	100%	1.29
med mobility low traffic	100%	0.98%	100%	0.47%
med mobility med traffic	99.96%	3.07%	100%	0.90%
med mobility high traffic	99.63%	4.93%	100%	1.30%
high mobility low traffic	99.91%	0.99%	100%	0.53%
high mobility med traffic	99.80%	3.39%	100%	0.85%
high mobility high traffic	99.96%	5.07%	100%	1.11%

There are also other advantages of our approach over other misuse-based and hand-coded approaches proposed in the literature. For example nodes monitor every packet and keep them in memory for a while in many approaches [17][20]. In addition they usually do it by using promiscuous monitoring which is expensive in terms of power usage. However the manual detection programs are simple, consist of only an if statement. They consume approximately 120 Wattch unit (power x cycles) energy. We also consider the effect of mobility on attacks in this research. Other approaches generally monitor if a node behaves properly or not without taking into account other factors affecting its behaviour.

So far we focus on developing an effective intrusion detection program for MANETs. However the limited resources available to many MANET nodes may affect proper working of intrusion detection systems. For instance, IDS agents might not be able to process every packet/alert due to limited resources. Efficiency is therefore an important factor for intrusion detection in mobile networks. We investigate this issue in the subsequent section.

7. Trade-offs in Intrusion Detection on MANETs

7.1. Related Work

Nodes on MANETs can vary from hand-held devices such as PDAs, cell phones, and the like to laptops that have different resource and computational capacities. Moreover they usually run on battery power. Currently proposed intrusion detection approaches for MANETs generally do not emphasise power consumption as an issue.

Some approaches propose a hierarchial intrusion detection architecture where the network is divided into small manageable groups such as clusters and zones. In this architecture some nodes have more IDS responsibility than others. For example, while all nodes are responsible for local intrusion detection in a cluster, clusterheads carry out global network-based intrusion detection. In some approaches [22][45] cluster heads are chosen based on criteria such as connectivity, remaining battery power, and the like. However, the selection mechanisms proposed for cluster-heads are neither investigated in detail nor implemented in those approaches. In other work such choices are made randomly for security reasons [46][47]. Using central management points to carry out computationally intensive tasks like data mining [48] for intrusion detection on MANETs has also been proposed.

The effect of limited resources (especially power) on intrusion detection on MANETs is rarely considered. In [49] monitoring nodes are voted and selected based on their connectivity and battery power. The weakness of this approach as stated in the paper [49] is that many control messages are sent for voting, since monitoring nodes need to be updated frequently. In this approach only monitoring nodes carry out intrusion detection, so a monitoring node listens to all traffic in its transmission range (not only that destined for itself), which is expensive in terms of power. In [50] an approach is proposed which determines network monitors based on available power in nodes. While all nodes carry out host monitoring, network monitoring is distributed to the nodes with sufficient power. More resource intensive detection algorithms can be applied to monitor critical nodes (such as the nodes at important positions in the network) as proposed in [51].

7.2. Analysis of Power Consumption of Evolved Programs

Firstly we analyse the power consumption of ad hoc flooding and route disruption detection programs evolved using GP. The same GP parameters and the same fitness function (equation 2) are used. To evaluate a program's energy consumption, we need to simulate the execution of the program. For that we firstly convert each individual (GP Tree) to a C program and write it to a file. In the transformation process from a GP tree

to a C program, the functions used by the individuals and not included in the standard C library (e.g. percent function) are defined as macros. After the C file is created, it is compiled and run on the Sim-Wattch simulator to simulate the execution of the program on PISA architecture and estimate its energy consumption. Wattch [52] is a framework for analyzing and optimizing microprocessor power dissipation for specific architectures. Sim-Wattch is a new simulator with the power models of Wattch integrated with the SimpleScalar architectural simulator [53].

The best individuals of ten runs with their energy consumptions are given in Figure 2. This figure shows that while classification accuracy is high, energy consumption of the program gets higher as well for ad hoc flooding attacks. It is much higher than the manual detection consumes. On the other hand, this relation is not quite straightforward for route disruption attacks. Analyzing the best individuals evolved for route disruption attacks shows that it is a simple attack can be detected by small programs which generally have a tendency to consume lower energy.

Furthermore, since the size of the programs can affect their energy consumption, we conduct experiments to evolve programs with different tree depths (17, 5). *Tree depth* defines the maximum size of the individuals (trees) evolved in GP. The effect of program size on evolved program's detection ability and energy consumption can be seen in Figure 2. It would appear that route discuption attacks can be detected by small programs. Constraining the sizes of program that can be evolved forces the technique to search for small programs in that reduced space. The approach manages to find effective ones. The results are more dramatic for ad hoc flooding detection. We can see how small-sized programs can exhibit good detection capability. Nevertheless programs with bigger program size and accordingly higher energy consumption show a slightly better performance on the detection ability.

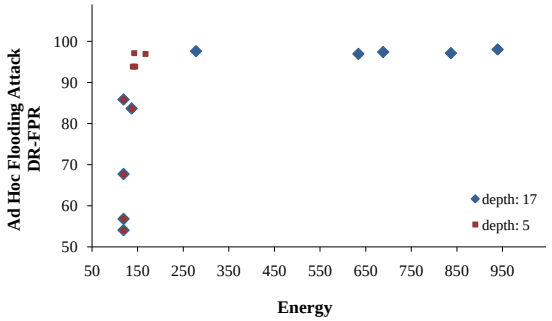

Figure 2: Classification Accuracy and Energy Consumption of the Optimal Evolved Programs

Bloat is a phenomenon whereby the size of individuals in a GP population increases dramatically over the duration of a run, largely due to redundant code [39]. The effect of bloat has also been noticed in our experiments; there are individuals with the same fitness but which have larger size due to code redundancy. It should be noted that small programs do not necessarily consume low energy, a program that uses expensive functions (such as multiply) could have higher energy consumption than a program with greater size.

These experiments demonstrate that different trade-offs can be made between classification accuracy and energy consumption of programs, and encourage us to find acceptable trade-offs between these objectives. A multi-objective evolutionary computation technique is employed to discover these trade-offs, as explained in the next section.

7.3. Discovering Trade-offs in Intrusion Detection Programs

Multi-objective evolutionary computation techniques are employed in order to optimise the following three objectives in our experiments: detection rate, false positive rate, and energy consumption of the program. The individual takes part in the evolution process and survives in the next generations based on its performance on these objectives.

$$f_1 = no.\ of\ attacks\ detected / no.\ of\ attacks \quad (3)$$

$$f_2 = 1 - no.\ of\ false\ events / no.\ of\ normal\ events \quad (4)$$

$$f_3 = 1 / energy\ consumption \quad (5)$$

The conceptual schema of the experiments is demonstrated in Figrure 3.

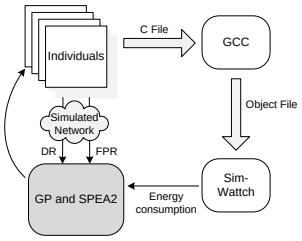

Figure 3: Simplified Schema of Experiments

7.3.1. Experiment 1: Attack-specific Intrusion Detection Programs

We firstly evolved intrusion detection programs for ad hoc flooding and route disruption attacks separately by using MOEC techniques. The same GP parameters are used except the population size (150) and SPEA2 archive size (100).

Figure 4 shows the optimal solutions found for ad hoc flooding attack at the 1000th generation. It shows the conditioning plots of detection rate (DR) versus false positive rate (FPR) which are produced conditional on the energy consumption of programs. Each chart shows the value DR and FPR of programs whose energy consumption falls in one of the intervals shown in the top of the figure. For example the top-right chart shows the programs whose energy consumption falls in the highest energy consumption and the bottom-left chart shows the programs with the lowest energy consumption. These charts show that the programs more close to the optimal solution (high detection rate and low false positive rate) consume higher energy consumption, as expected. In programs with lower energy consumption the false positive rate increases noticably. Overall,

the trade-offs falling in the second biggest energy consumption interval (the chart in the top-middle) could be more optimal and acceptable, since it include points which achieve the high classification accuracy with less energy consumption (than the programs consume in the interval f).

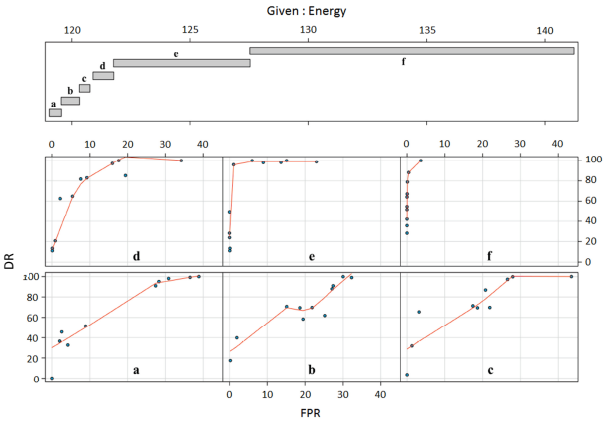

Figure 4: Coplots for Programs Evolved for Detection of Ad Hoc Flooding Attack with Three Objectives

For the route disruption attack, programs closer to the optimum solution which have higher detection ability and lower energy consumption are achieved by using MOEC techniques. We have compared energy consumption of programs which have a high-accuracy detection ability with the programs evolved using GP. It is observed that programs with lower energy consumption stand out in the results obtained by MOEC techniques, particularly for the ad hoc flooding attack, energy consumption is significantly reduced. It could also be the effect of reduced bloat by using SPEA2.

7.3.2. Experiment 2: Evolving Multi-attack Intrusion Detection Programs

In this part we evolve programs to detect ad hoc flooding and route disruption attacks together by using MOEC techniques. We aim to investigate if it is better to evolve one program to detect both attacks or evolve two programs each using half the resource usage. The following multi-fitness function which evaluates the performance of program on both attacks, and its energy consumption is employed.

$$f_1 = \frac{(detection\ rate_{flooding} + detection\ rate_{r.disruption})}{2} \quad (6)$$

$$f_2 = 1 - \frac{(false\ positive\ rate_{flooding} + false\ positive\ rate_{r.disruption})}{2} \quad (7)$$

$$f_3 = 1/energy\ consumption \quad (8)$$

Figure 5 shows the conditioning plot diagrams for three objectives on detection of both attacks together. The programs with lower false positive rate generally have higher energy consumption. However some good results (in terms of classification accuracy) are also observed in the lower energy consumption intervals. Overall the results demonstrate that a detection

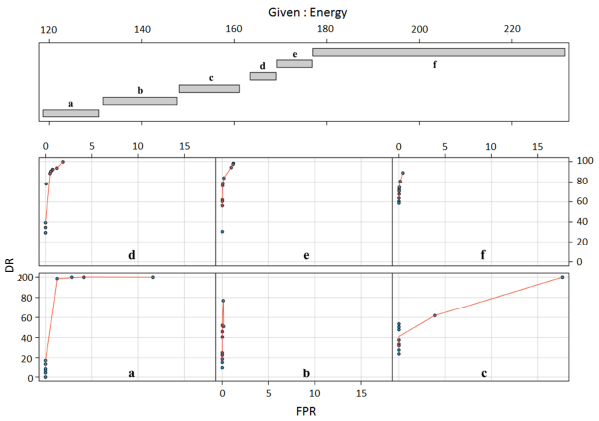

Figure 5: Coplots for Programs Evolved for Detection of Both Attacks Together with Three Objectives

program for both attacks can be more energy-efficient than two programs which detect these attacks separately, although it does not show as high classification accuracy as the two programs do separately. In the results of thirty runs, there is no program evolved for detecting both attacks which has false positive rate less than 2% (with high detection rate) simultaneously. It is believed that evolving programs to detect similar attacks (showing similar consequences on the network) such as DoS attacks together could give more promising results in terms of energy consumption than evolving separate programs for each attack, since they are more likely to have similar signatures.

The performance of some evolved programs (detecting both attacks together) is demonstrated in Table 5.

Table 5: Performance of Some Programs for Detection of Both Attacks Together

Flooding Attack		**Route Disruption Attack**		**Resource (Wattch Units)**
DR	FPR	DR	FPR	
98.80%	1.90%	100%	2.97	≃131
97.71%	1.75%	100%	2.48	≃169
98.80%	1.86%	97.78%	2.90	≃174

These experiments show that different trade-offs could be made between classification accuracy and energy consumption. It is shown how in some circumstances a multiple objective approach provides a more effective means of searching the trade-off space. It is likely that for some types of networks (*e.g.* sensor networks) the ability to make good trade-offs will be particularly important. Programs with almost the same classification accuracy (as obtained by GP) but with lower energy consumption are evolved by using MOEC techniques. More importantly a set of solutions showing different trade-offs is obtained. So programs showing different trade-offs could be distributed to nodes based on their energy level. A final choice between solutions making different trade-offs rests with the designer.

Our work is unusual in that it trades off security performance (detection and false positive rates) against resources (power).

The inherent complexity of MANET operations makes it difficult to see how IDS programs with optimal trade-offs could be obtained by standard system development practices. An optimisation based approach seems a natural and effective candidate for this problem.

8. Distributed and Cooperative Intrusion Detection

8.1. Introduction

So far it is assumed that each node in the network carries out monitoring and detecting of malicious activities. Every node has an intrusion detection agent and detects network attacks locally. Hence the evolved programs are distributed to each node on the network. In this section further intrusion detection architectures suited to this distributed and resource-constrained environment are explored.

Intrusion detection architectures on MANETs can be classified into two main groups: local detection, and distributed and coopeative detection. In local detection, every node in the network has an IDS agent and detects attacks on its own without collaborating with other nodes. Hence the evolved programs are distributed to each node. However a node on a MANET can only see a portion of the network: the packets in its radio range and the packets which it sends or receives. Network attacks (network scans, distributed attacks, etc.) cannot be detected with the partial network data held by local nodes in this environment. Furthermore, in practice, limited resource availability may affect the ability of nodes to engage in intrusion detection.

In a distributed and cooperative architecture each node has IDS agents as in the local detection architecture, but they can also communicate with other nodes to exchange information, to reach decisions and to agree on responses. One of the most used cooperative detection architecture in the literature is based on hierarchy among nodes. The biggest drawback of this architecture is the high cost of building and maintaining the hierarchy in a highly dynamic environment. Message sending and receiving is also very expensive in terms of energy consumption. Communication between these IDS agents is provided either by exchanging data directly or by use of mobile agents.

Here we explore an intrusion detection architecture suited to the distributed and resource-constrained environments of interest. Two intrusion detection architectures are investigated in this research: local detection and cooperative detection in the neighbourhood. A distributed and cooperative intrusion detection architecture where nodes communicate with their immediate (one-hop away) neighbours to reach decisions is proposed. The following question is explored: "Is it possible to increase the effectiveness of an agent's intrusion detection by collaboration with the IDS agents in its neighbourhood?". The efficiency of this architecture in terms of bandwidth and energy usage is also explored and compared with that of local detection.

8.2. Cooperative Detection in Neighbourhood by GE

We have seen that ad hoc flooding attacks can be detected effectively by programs synthesised using evolutionary computation techniques in Section 6.1. By nature an ad hoc flooding attack is a distributed DoS attack and floods almost every node in the network with broadcast RREQ packets. When a node believes that someone is attacking (flooding) him, he can support his judgement with the help of his neighbour nodes since he gets RREQ packets through them. Accordingly we believe that cooperative intrusion detection with neighbour nodes can increase the fitness value. So we extend the experiments to evolve a distributed and cooperative detection program.

8.2.1. Extending the Grammar for Cooperative Detection

The grammar introduced in Table 2 is extended in order to evolve a cooperative detection program for ad hoc flooding attacks. The main difference here is that it allows using features from neighbouring nodes to make a decision. Each neighbour executes the evolved statement and sends its result to the main node. These results from each neighbour are combined at this node. Moreover while the local detection grammar returns an if statement, the cooperative detection grammar returns if or if-else statement (*Cooperative Detection 1*) as shown in the following. The logic behind this is that we can evolve programs for each node to detect attacks locally where we can, and ask more information from its neighbour nodes otherwise. Other primitives used in the grammar are the same.

Local Detection:
if condition satisfied satisfied **then**
  raise alarm
end if

Cooperative Detection-1:
if condition satisfied satisfied **then**
  {locally or cooperatively}
  raise alarm
else
  Local Detection or Cooperative Detection-1
end if

The GE algorithm is run thirty times. The same GE parameters except the generations(=4000) are used. Since the grammar is more complex, we increase the number of generations until termination. The performance of the best program evolved is demonstrated in Table 6. Detection rate and false positive rates are used to evaluate its performance. It is also compared with the best local detection program evolved in Section 6.1. The results show that cooperative detection algorithm (*Cooperative Detection 1*) can achieve a lower false positive ratio (0.06%-0.31%) compared to systems employing local detection only in these experiments.

All the best evolved programs of each run return an if statement which uses information only from neighbour nodes. However, a program that detects attacks locally where it can is more desirable to preserve limited bandwidth. Furthermore, message sending and receiving is very expensive in terms of energy consumption. It is observed that the grammatical evolution algorithm tends to evolve simplified programs. For that reason GE algorithm is forced to evolve a program in the structure *Cooperative Detection 2* by changing the grammar.

Cooperative Detection-2:

if condition satisfied satisfied **then**
{suspect locally that there might be a malicious activity happening}
if condition satisfied satisfied **then**
{reach on a decision based on the further information obtained from neighbour nodes}
raise alarm
end if
end if

GE is run thirty times using the new grammar (*Cooperative Detection 2*). The results show that the false positive ratio is significantly reduced. Furthermore the communnication workload between nodes is reduced approximately 70% compared to the best evolved program with the grammar *Cooperative Detection 1* where a node asks for information from its neighbours at each time interval. However the best program evolved below by the grammar *Cooperative Detection 2* consults its neighbour nodes only when it believes there is a chance of malicious activity on the network.

Table 6: Comparison of Local and Cooperative Intrusion Detection Programs Evolved by GE for Detection of Ad Hoc Flooding Attack

Network	Local Detection		Coop. Detection-1		Coop. Detection-2	
	DR	FPR	DR	FPR	DR	FPR
low mobility low traffic	99.81%	0.29%	99.62%	0.23%	99.29%	0.20%
low mobility medium traffic	98.54%	1.72%	99.48%	1.58%	98.65%	1.43%
low mobility high traffic	98.14%	1.99%	97.84%	1.59%	97.50%	1.52%
medium mobility low traffic	99.86%	0.36%	99.86%	0.22%	99.86%	0.28%
medium mobility medium traffic	99.86%	2.00%	99.61%	1.69%	99.44%	1.32%
medium mobility high traffic	98.59%	2.79%	98.23%	2.31%	97.43%	2.18%
high mobility low traffic	99.96%	0.66%	99.49%	0.60%	99.32%	0.47%
high mobility medium traffic	98.66%	1.73%	98.15%	1.42%	97.61%	1.32%
high mobility high traffic	99.65%	2.91%	99.47%	2.42%	99.25%	2.17%

We here present the performance of the program with the minimum false positive rate. The program decreases the false positive rate with a small decrease in detection rate. Different programs with different trade-offs between detection rate and false positive rate are also seen in the results.

In conclusion, cooperative intrusion detection programs which achieve lower false positive rates than local detection are evolved by using GE in these runs. Furthermore the interaction between IDS agents is investigated and is reduced enormously by improving the BNF grammar for the problem. GE provides a great flexibility in changing the representation of a problem by changing the BNF grammar easily. The best programs evolved with each grammar are shown in Table 7.

8.3. Investigating the Resource Usage of Cooperative Detection Programs

In this section, we evaluate the energy usage of cooperative intrusion detection programs and compare it with the energy usage of local detection programs, since the suitability of these

Table 7: The Programs –Best Individuals– Evolved by GE for Detection of Ad Hoc Flooding Attack

Detection Type	Evolved Program
Local Detection	if((send_rrepPs + exp(frw_aodvPs - updated_routes * pow(frw_rreqPs, added_repairedroutes))) > no_neighbours)
Coop. Detection-1	if((abs(floor(frw_aodvPs) - send_rreqPs) - log(no_neighbours)) > (updated_routes)) //by neighbour nodes
Coop. Detection-2	if(invroutes_timeout < frw_rreqPs && added_neighbours < frw_aodvPs) if((frw_aodvPs - updated_routes) > (max(exp(0.93) + 0.21 - addedroutes_notice, 0.20)) //by neighbour nodes

evolved programs to MANETs is crucial. The trade-offs between intrusion detection ability and resource consumption of programs (in terms of energy and bandwidth) are discovered.

The SPEA2 algorithm is run to evolve power and bandwidth efficient intrusion detection programs. The algorithm aims to maximize the following objectives simultaneously.

$$f_1 = no.\ of\ attacks\ detected / no.\ of\ attacks \tag{9}$$
$$f_2 = 1 - no.\ of\ false\ events / no.\ of\ normal\ events \tag{10}$$
$$f_3 = 1 / energy\ consumption \tag{11}$$
$$f_4 = 1\ -\ \%\ of\ neighbour\ nodes\ in\ cooperation \tag{12}$$

In [54] energy consumption in ad hoc networks is modelled for the four states (transmit, receive, sleep, and idle) of the network interface. The cost to a node to send or receive a packet on the network layer is modeled by the linear equation below. The cost associated with channel acquisition (m and b) is assumed to be fixed. The cost of a sending/receiving packet is proportional to the size of the packet. If it is a point-to-point traffic, the total cost is calculated as follows.

$$Cost = m_{send} \times size + b_{send} \tag{13}$$
$$+ m_{recv} \times size + b_{recv} \tag{14}$$

The packet size (the output of evolved programs) in our experiments is quite small, generally it is one or two floating point numbers. However each neighbour node participates in detection by sending their local information. More cooperating neighbours means more communication and more energy/bandwidth consumption. Accordingly we seek to reduce it as far as is practical. A simple way of achieving this is to seek to reduce the number of immediate neighbouring nodes that a node collaborates with to reach an IDS decision. Therefore f_4 is added to the multi-fitness function. The trade-offs among the number of neighbours in cooperation, detection ability and power usage of evolved programs are to be discovered.

In mobile networks radio communication has very high power consumption. A node consumes an amount of power (=Rx receiving power) even in its idle state in order to monitor the channel. While transmission power (=Tx) is higher, it depends on the transmission range. A node should maintain its transmission power at a level sufficient to reach receiving nodes, or switch to a sleep state to save its power locally. Researchers generally focus on reducing the communication power since this is the main cause of battery depletion. In this research even

though an intrusion detection program is likely to consume a small amount of power compared to the communication power, it runs continuously. When the communication is inevitable in mobile networks, controlling the power usage of programs running on a node is another approach to save battery power locally. The degree to which a node engages in "passive" monitoring may also be a factor. With largely passive operation, the relative importance of power consumption of programs increases. These are the main reasons that we consider the power consumed by intrusion detection programs besides its communication power in this research. However, we are aware that further integration of these concepts is also possible. For example, we could reduce the rate at which neihgbouring information is requested (and hence save power) but this would inevitably give rise to a lower detection capability. (That is to say there is a trade-off to be made here.)

8.4. Results Analysis

SPEA2 was run thirty times. Each run produced a Pareto optimal set of non-dominated solutions. We analyse the distribution of the union of those thirty sets. (Within the union some individual solutions from one contributing set may dominate or be dominated by individual solutions from another contributing set). We really want solutions that work well on all three criteria. Consider, for example, the set of solutions with (f_1+f_2-1) >= 0.8, f_3 >= 0.7 and f_4 >= 0.5. Solutions in this set clearly perform well on all three axes. In some respects we may consider them "excellent", or an acceptable outcome from a run of our technique. The average number of "acceptable" solutions in each run is 2.8333 (with a standard deviation of 2.0356). The number of runs which produced no "acceptable" solution is 6. Of course, different definitions of acceptable are clearly possible.

MOEC returns a set of solutions and one solution has four values showing different trade-offs between the multi-objectives described in Equation 9-12. Since analyzing the multi-dimensonal data is difficult, we show the relation between each pair of multi-objectives separately here. Only programs with high classification accuracy (where the of detection rate and (100-false positive rate) are bigger than 98%) are selected here and evaluated on a network under medium mobility and medium traffic.

Firsly, the relation between the fitness values of these programs and the percentage of neighbour nodes asked for information to achieve these fitness values is analysed. It is observed that when the number of neighbours participating in the detection increases, the effectiveness of the programs increases as well. The correlation between these values is analyzed for each run and the average correlation coefficient is calculated as 0.3829∓0.0963. This value (with the p-value 0.0142∓0.0226) shows that these two values are correlated, however it is not a very strong relationship. There are other influences and the relationship between these values is not fixed. Since the aim is to find different trade-offs among four objectives, the value of one objective is affected by other objectives as well.

There is no statistifically significant correlation found between energy usage of programs and the percentage of neighbour nodes in cooperation. Programs with higher classification accuracy tend to consume more energy. We here present a few solutions produced in one run. These values are weakly correlated in some runs where the average correlation coefficient is 0.3070∓0.0131 (with the p-value 0.0167∓0.0038). It means while the energy increases, the fitness increases too in the output non-dominated solutions. However the fitness is also affected by other objectives. Moreover it is observed that the energy consumption of these programs is lower than that of the programs given in the previous sections which only considers classification accuracy as the fitness function.

The performance of some of these programs evolved is demonstrated in Table 8. The best classification accuracy is achieved by program 4 which uses 80% of neighbour nodes to reach a decision. Program 1 consumes higher energy than the program 4, but decreases the number of nodes participating in intrusion detection very significantly. Its detection ability (fitness value) is as almost the same as that of local detection and it uses half of its neighbour nodes. It decreases the false positive rate with a small decrease in the detection rate. The energy consumption of evolved programs is generally almost the same as for the power-aware local detection programs' evolved in Section 7.3. Different trade-offs among the classification accuracy, the energy consumption and the number of neighbour nodes in cooperation are clearly seen in these results. Here we are using the number of collaborating nodes as a proxy for resource consumption incurred by collaboration. Thus, we need to send requests to our neighbours and they must respond. This incurs both broadcast and reception costs but also information retrieval costs within each of the neighbouring nodes. It is clear that we should seek to reduce the number of collaborating neighbours as much as it is practical. In this approach neighbours with enough resources could participate in intrusion detection.

Table 8: Performance of Some Cooperative Programs Evolved by MOEC

Program No.	Detection Rate	False Positive Rate	Nodes	Energy
1	99.21%	1.32%	51%	≃ 149
2	99.53%	1.54%	58%	≃ 151
3	99.65%	1.72%	74%	≃ 149
4	99.41%	1.23%	80%	≃ 137

The performance of evolved programs is demonstrated on networks only under medium mobility and medium traffic as shown in Table 8. On simulated networks under high mobility the detection rate is decreased down to 97% with a decrease in the false positive rate down to 1%.

This section demonstrates the potential use of evolutionary computation techniques to discover complex properties of MANETs (such as limited power and limited bandwidth) and to generate a suitable intrusion detection approach applicable to this new environment. It is shown that cooperative intrusion detection with neighbour nodes increase the effectiveness of the system. The energy consumption of these programs is almost the same as that of the power-aware local detection programs evolved in Section 7.3. However cooperative intrusion detection requires communication between nodes. Even though the size of the packets sent for intrusion detection is small, message

sending and receiving still consume an amount of energy and the number of neigbour nodes taking a role in intrusion detection is a legitimate target for minimization. The results show that the performance of the local detection programs can be achieved by using a distributed and cooperative detection program. Some neighbouring nodes (50% of neighbouring nodes asked for information) participate in cooperative intrusion detection. If the number of nodes in cooperation increases, the classification accuracy increases as well. Different trade-offs that can be applied according to the application are presented here. We believe this is one of the first works to consider constrained resources in the case of IDS for MANETs.

9. The Evaluation of GP and GE on Intrusion Detection

Two variants of evolutionary computation techniques, namely Genetic Programming (GP) and Grammatical Evolution (GE), have been evaluated to design intrusion detection programs for known attacks against MANETs so far. It is shown that GP and GE are good at discovering complex relations on MANETs. In these experiments the default parameters (of ECJ [42] and libGE [44]) are used for each technique. However each technique shows different performance under different parameter settings. In this section each technique is analysed using their approximate optimal parameters for an unbiased evaluation.

Here we aim to identify approximations to the optimal parameters for GP and GE and, then make a fair comparison of these techniques under their optimal parameter settings. We follow the steps of a simple Design of Experiments (DoE) methodology in [55].

Genetic Programming and Grammatical Evolution differ from each other in two fundamental ways [40]. The main difference is the representation of individuals. While individuals are represented as trees in GP, GE employs linear genomes. GE also performs mapping from genotype to phenotype (program) and uses a grammar to dictate legal structures in the phenotypic space [40]. We aim to show if these differences make a statistically significant difference on the performance of detection programs evolved by using these techniques separately.

The experimental results so far show that a route disruption attack can be detected easily since it violates the specifications of the routing protocol directly. However, ad hoc flooding is a more complex attack and not easily differentiable from normal behaviour of the system due to the dynamic nature of MANETs. It should provide a challenging test case of our GP and GE approaches.

9.1. The Testbed

Firstly, a common platform is built for each evolutionary computation algorithm before tuning the parameters. Each algorithm uses the feature set in Table 1 and, the same training and testing data simulated by ns2 [33] and BonnMotion [34]. The functions used to define the problem in a GP tree are those in the GE grammar. Strongly-typed GP is employed here to enforce the rules of the GE grammar into the GP. It also helps providing the code ordering as in the GE grammar. For example, only relational functions could be allowed to be placed to the first level of the GP tree. Then we could put the relational functions into an if-statement while we are translating the individual tree to a C program. Consequently, it provides an output program returning an if-statement as in the GE grammar. Hence, the programs in the same format could be evolved using either GP or GE.

The size of an individual is constrained by tree depth in GP, and by genome size and wrapping in GE. Maximum tree depth size is one of the important parameters in GP. Maximum genome size in GE, corresponding to maximum tree depth in GP, is also defined in our experiments to evolve individuals in the same size range in each technique. Maximum genome size is computed by building the full tree with maximum tree depth and defining the grammar which builds this tree intuitively. Since our grammar is not complicated, it is estimated easily. However both algorithms behave differently when the size of an individual exceeds the predefined parameter or an invalid individual is created in general. While GP copies the parents of the invalid individual after retries of creating a valid individual, GE assigns the lowest fitness value to the individual.

Koza's ramped half and half initialization in GP and sensible initialization in GE are used to create an initial population. GE sensible initialization is based on Ramped Half and Half Initialization in GP but generates derivation trees of equivalent size [40].

In the experiments, tournament selection was used to select individuals for recombination and simple genetic algorithm was used to select individuals for replacement. Furthermore elitist approach were employed in each technique by keeping the best individual of each population. Necessary changes were made to each library in order to implement these algorithms in a standard way.

The details of other parameters used/tuned in our experiments are given in the subsequent section.

9.2. The Design of Experiments

In this methodology we firstly tune the parameters to identify approximations to the optimal parameters for each technique. However finding the optimal parameters for an algorithm requires a large-scale experimentation consuming vast computing resources. So we take the approach which assumes "if an equivalent amount of effort is spent in applying this method to each of the algorithms, it is reasonable to expect the approximations to be similarly close to the absolute optimum for each algorithm and so the comparisons to be fair" [55]. Therefore we start with finding approximations to the optimal parameters for each technique in our research.

In our experiments we use four independent parameters : crossover probability [0.1, 0.9], mutation probability [0.01, 0.5], population size [50, 100], and tournament size [2, 9]. Running experiments at each possible parameter setting is computationally infeasible, so we use three-level full factorial design where each parameter is considered at three levels, referred as low, intermediate and high levels.

We chose the ranges large enough to cover all practical values for each parameter. For crossover probability the highest value

is chosen as 0.9 since leaving some part of old population survive to next generation is believed to be good [56]. Mutation helps to avoid being trapped in local extremes by introducing diversity into the system. However it should not occur very often, otherwise genetic algorithms can simply return to random search [56]. That is the reason the highest value for mutation probability is chosen as 0.5 here. Our pilot study also supports that the algrotihms do not show better performance under high mutation probability. The parameter for termination criteria of an evolutionary algorithm namely *maximum number of generations* (100.000 / population size) depends on the parameter *population size* to ensure the same number of individuals created in overall in each algorithm. Each algorithm is run twice (with different seeds) for each parameter setting. Hence each algorithm run totally 3^4x2 times to estimate β coefficients in the second linear model below which describes the relationship between the performance of the algorithm, y, and the parameter settings, x_i.

$$y = \beta_0 + \sum_i \beta_i x_i + \sum_i \sum_{j>i} \beta_{ij} x_i x_j \sum_i \beta_{ii} x_i^2 + \epsilon \quad (15)$$

After estimating the coefficents in the linear model by using a standard linear regression, we apply quadratic programming to locate approximations to the optimal parameters. The parameter settings (crossover probability, mutation probability, tournament size, population size, maximum number of generations) which give the best performance are found (0.1, 0.37, 8, 50, 2000) for GP and (0.9, 0.5, 7, 1000, 100) for GE.

Finally each algorithm is run one hundred times at the optimal parameter settings found and compared by applying a statistical hypothesis test of equality. The results show that the mean of GP runs (fitness values) is greater than the mean of GE runs with 95% confidence. However better results by GE which uses steady-state approach have been observed in the experiments presented in Section 6.1. This approach is proposed to reduce the effect of invalid individuals in GE. That is why the same methodology is applied to compare two techniques with the same parameter values, but with a steady-state approach. The standard steady-state approach which replaces the worst individual of the preceeding population is employed. Even if this individual is better than the new individual, it will be replaced regardless of its better score. The parameter settings (crossover probability, mutation probability, tournament size, population size, maximum number of generations) which give the best performance are found (0.1, 0.01, 9, 50, 2000) for GP and (0.9, 0.5, 2, 514, 194) for GE.

After running each algorithm one hundred times with the optimal parameter settings, hypothesis testing is applied for comparison. The results show that if a steady-state approach is employed, the mean of GE runs (fitness values) is greater than the mean of GP runs with 90% confidence. It is reasonable to say that it was invalid individuals who affect the performance of GE (slow down the evolution) in the simple approach. Overall when the results of the best GP version (with simple approach) is compared with the results of the best GE version (with steady-state approach), GP shows a better performance.

10. Conclusions

Many IDSs have been proposed for conventional networks, but they are not naturally applicable to MANETs. The lack of central points, significant node mobility, and limited resource availability pose significant challenges to IDS. In this paper we have investigated the use of evolutionary computation techniques to discover detectors suited to such complex environments. We have applied grammatical evolution and genetic programming techniques to detect ad hoc flooding and route disruption attacks on AODV. The evolved programs show good performance on simulated networks with varying mobility and traffic patterns.

Efficiency is an important criterion for MANETs' highly resource-constrained nodes. We have shown how energy usage and detection ability can be traded off and that evolutionary techniques have very significant potential to explore the complex IDS program design space. The classic notion of Pareto fronts provides a principled means to compare the trade-offs involved. Our paper presents the first multi-objective optimisation approach to trading off multiple criteria in MANET IDS construction, and the first to trade off functional and non-functional properties. Our focus on power usage will find application elsewhere. For example, many ad hoc self-organising sensor networks (e.g. such as those formed by smart dust networks) will be severely resource constrained. Finally, we have shown how evolutionary techniques can be used to evolve IDSs with both local and cooperative architectures.

Our simulation based approach to fitness evaluation (both detection ability and power usage) provides an excellent practical means for exploring what are clearly highly complex design spaces. Our mobility model (provided by BonnMotion[34]) is a simple one. Alternative mobility models should now be investigated. However, there is no reason in principle why fitness could not be evaluated over several mobility models simultaneously. Indeed, this would largely be a question of computational cost. One could even imagine considering detection abilities and power consumptions over different mobility models as different criteria to be traded off.

Overall, we have shown that evolutionary techniques such as genetic porogramming, grammatical evolution and multi-objective evolutionary algorithms have considerable potential for exploring the complex design spaces associated with IDS on MANETs and are capable of producing high performing programs that take various functional and non-functional criteria into account. We know of no other approach that can do this.

References

[1] Y. Li, J. Wei, Guidelines on selecting intrusion detection methods in manet, in: Proceedings of the Information Systems Education Conference, EDSIG, 2004, pp. 1–17.

[2] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, L. Viennot, Optimized link state routing protocol for ad hoc networks, in: Multi Topic Conference, IEEE INMIC. Technology for the 21st Century, IEEE, 2001, pp. 62 – 68.

[3] C. Perkins, E. Royer, Ad-hoc on-demand distance vector routing, in: Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications, IEEE, 1999, pp. 90–100.

[4] D. B. Johnson, D. A. Maltz, Dynamic source routing in ad hoc wireless networks, in: Mobile Computing, Kluwer Academic Publishers, 1996, pp. 153–181.

[5] D. Denning, An intrusion detection model, IEEE Transactions on Software Engineering 13 (1987) 222–232.

[6] P. Uppuluri, R. Sekar, Experiences with specification-based intrusion detection, in: Proceedings of the Recent Advances in Intrusion Detection (RAID), LNCS 2212, Springer, 2001, pp. 172–189.

[7] M. Crosbie, G. Stafford, Applying genetic programming to intrusion detection, in: Proceedings of AAAI Symposium on Genetic Programming, Cambridge, MA, 1995, pp. 1–8.

[8] L. Me, Gassata: A genetic algorithm as an alternative tool for security audit trails analysis, in: Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID), Springer, 1998, pp. 1–11.

[9] W. Li, Using genetic algorithm for network intrusion detection, in: Proceedings of the United States Department of Energy Cyber Security Training Conference, USA, 2004, pp. 24–27.

[10] KDD cup 1999 intrusion detection data set, 2010.

[11] A. Abraham, C. Grosan, C. Martiv-Vide, Evolutionary design of intrusion detection programs, International Journal of Network Security 4 (2007) 328–339.

[12] A. Abraham, C. Grosan, Evolving intrusion detection systems, in: Genetic Systems Programming: Theory and Experiences, volume 13, Springer, 2006, pp. 57–79.

[13] D. Song, M. I. Heywood, A. N. Zincir-Heywood, Training genetic programming on half a million patterns: An example from anomaly detection, IEEE Transactions on Evolutionary Computation 9 (2005).

[14] D. Wilson, D. Kaur, Knowledge extraction from kdd'99 intrusion data using grammatical evolution, WSEAS Transactions on Information Science and Applications 4 (2007) 237–244.

[15] C.-Y. Tseng, P. Balasubramayan, C. Ko, R. Limprasittiporn, J. Rowe, K. Lewitt, A specification-based intrusion detection system for aodv, in: Proceedings of the ACM Workshop on Security in Ad Hoc and Sensor Networks, ACM, 2003, pp. 125–134.

[16] C. Tseng, S.-H. Wang, W. Lee, C. Ko, K. Lewitt, Demem: Distributed evidence driven message exchange intrusion detection model for manet, in: Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID), Springer, 2006, pp. 249–271.

[17] G. Vigna, S. Gwalani, K. Srinivasan, E. M. Belding-Royer, R. A. Kemmerer, An intrusion detection tool for aodv-based ad hoc wireless networks, in: Proceedings of the 20th Annual Computer Security Applications Conference, IEEE Computer Society, 2004, pp. 16–27.

[18] J. Parker, J. Undercoffer, J. Pinkston, A. Joshi, On intrusion detection and response for mobile ad hoc networks, in: Proceedings of the 23th IEEE Int. Performance Computing and Communications Conference, IEEE, 2004.

[19] S. Buchegger, J.-Y. L. Boudec, Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks, in: Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, IEEE Computer Society, 2002, pp. 403–410.

[20] S. Marti, T. J. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in: Proceedings of ACM Int. Conf. on Mobile Computing and Networking, ACM, 2000, pp. 255–265.

[21] F. Anjum, R. Talpade, Lipad: lightweight packet drop detection for ad hoc networks, in: Proceedings of the 60th IEEE Vehicular Technology Conference, IEEE, 2004, pp. 1233–1237.

[22] O. Kachirski, R. Guha, Effective intrusion detection usign multiple sensors in wireless ad hoc networks, in: Proceedings of the 36th IEEE International Conference on System Sciences, IEEE, 2003.

[23] Y. Zhang, W. Lee, Y. an Huang, Intrusion detection techniques for mobile wireless networks, Wireless Networks Journal (ACM WINET) 2 (2003).

[24] B. Sun, K. Wu, U. Pooch, Zone-based intrusion detection for mobile ad hoc networks, International Journal of Ad Hoc and Sensor Wireless Networks 2 (2003).

[25] Y. Huang, W. Lee, A cooperative intrusion detection system for ad hoc networks, in: Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks, ACM, 2003.

[26] S. Sen, J. A. Clark, Intrusion detection in mobile ad hoc networks, in: Guide to Wireless Ad Hoc Networks, Springer, 2009, pp. 427–454.

[27] T. Weise, Genetic Programming for Sensor Networks, Technical Report, 2006.

[28] D. M. Johnson, A. M. Teredesai, R. T. Saltarelli, Genetic programming in wireless sensor networks, in: Proceedings of the European Conference on Genetic Programming (EUROGP), LNCS 3447, Springer, 2005, pp. 96–107.

[29] S. Sen, J. A. Clark, A grammatical evolution approach to intrusion detection on mobile ad hoc networks, in: Proceedings of the Second ACM Conference on Wireless Network Security, ACM, 2009, pp. 95–102.

[30] S. Sen, J. A. Clark, J. E. Tapiador, Power-aware intrusion detection in mobile ad hoc networks, in: Proceedings of the International ICST Conference on Ad Hoc Networks, Springer, 2009, pp. 224–239.

[31] S. Sen, J. A. Clark, J. E. Tapiador, Security threats in mobile ad hoc networks, in: Security of Self-Organizing Networks: MANET, WSN, WMN, VANET, Auerbach Publications, 2010, pp. 127–147.

[32] P. Ning, K. Sun, How to Misuse AODV: A Case Study of Insider Attacks against Mobile Ad Hoc Routing Protocols, Technical Report, Department of Computer Science, North Caroline State University, 2003.

[33] Ns-2: The network simulator, 2010.

[34] Bonnmotion: A mobility scenario generation and analysis tool, 2010.

[35] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research, Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications 2 (2002) 483–502.

[36] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, Springer, 2003.

[37] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

[38] W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francome, Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufman Publishers, 1998.

[39] C. Ryan, J. Colline, M. O'Neill, Grammatical evolution: Evolving programs for an arbitrary language, in: Proceedings of the 1st European Workshop on Genetic Programming, LNCS 1391, Springer, 1998, pp. 83–95.

[40] M. O'Neill, C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language, Springer, 2003.

[41] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report 103, Swiss Federal Institute of Technology, 2001.

[42] Ecj18: A java-based evolutionary computation research system, 2010.

[43] D. J. Montana, Strongly typed genetic programming, Evolutionary Computation 3 (1995) 199–230.

[44] libge: A C++ library for grammatical evolution, 2010. Http://bds.ul.ie/libGE.

[45] D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng, T. Bowen, A general cooperative intrusion detection architecture for manets, in: Proceedings of the 3rd International Workshop on Information Assurance, IEEE, 2005, pp. 57–70.

[46] Y. Huang, W. Fan, W. Lee, P. S. Yu, Cross-feature analysis for detection ad-hoc routing anomalies, in: Proceedings of the 23rd International Conference on Distributed Computing Systems, IEEE, 2003, pp. 478–487.

[47] P. Yi, Y. Zhong, S. Zhang, A novel intrusion detection method for mobile ad hoc networks, in: Proceedings of Advances in Grid Computing (EGC), LNCS 3470, Springer, 2005, pp. 1183–1192.

[48] A. Smith, An examination of an intrusion detection architecture for wireless ad hoc networks, in: Proceedings of the 5th National Colloquium for Information System Security Education, USA, 2001.

[49] H. Kim, D. Kim, S. Kim, Lifetime-enhancing selection of monitoring nodes for intrusion detection in mobile ad hoc networks, International Journal of Electronics and Communications 60 (2006) 248–250.

[50] T. Srinivasan, V. Mahadevan, A. Meyyappan, A. Manikandan, M. Nivedita, N. Pavithra, Hybrid agents for power-aware intrusion detection in highly mobile ad hoc networks, in: Proceedings of the International Conference on Systems and Network Communication, IEEE Computer Society, 2006.

[51] A. Karygiannis, E. Antonakakis, A. Apostolopoulos, Detecting critical nodes for MANET intrusion detection systems, in: Proceedings of the 2nd International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing, IEEE, 2006.

[52] D. Brooks, V. Tiwari, M. Martonosi, Wattch: A framework for architectural-level power analysis and optimizations, in: Proceedings

of the 27th International Symposiyum on Computer Architecture, IEEE, 2000.
[53] Simplescalar, 2010. Http://www.simplescalar.com/.
[54] L. M. Feeney, An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks, Mobile Networks and Applications 6 (2001) 239–249.
[55] D. R. White, S. Poulding, A rigorous evaluation of crossover and mutation in genetic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference (EuroGP), LNCS 5481, Springer, 2009, pp. 220–231.
[56] S. Sivanandam, S. Deepa, Introduction to Genetic Algorithms, Springer, 2008.