The ISC Int'l Journal of Information Security

ISeCure

January 2012, Volume 4, Number 1 (pp. 25–39)

http://www.isecure-journal.org

BeeID: Intrusion Detection in AODV-based MANETs Using Artificial Bee Colony and Negative Selection Algorithms☆

Fatemeh Barani [1] and Mahdi Abadi [1,*]

[1] *Faculty of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran*

ARTICLE INFO.

Article history:
Received: 14 November 2011
Revised: 30 January 2012
Accepted: 31 January 2012
Published Online: 30 May 2012

Keywords:
Mobile Ad Hoc Network, Routing Attack, Intrusion Detection, Artificial Bee Colony, Negative Selection, Monte Carlo Integration.

ABSTRACT

Mobile ad hoc networks (MANETs) are multi-hop wireless networks of mobile nodes constructed dynamically without the use of any fixed network infrastructure. Due to inherent characteristics of these networks, malicious nodes can easily disrupt the routing process. A traditional approach to detect such malicious network activities is to build a profile of the normal network traffic, and then identify an activity as suspicious if it deviates from this profile. As the topology of a MANET constantly changes over time, the simple use of a static profile is not efficient. In this paper, we present a dynamic hybrid approach based on the artificial bee colony (ABC) and negative selection (NS) algorithms, called *BeeID*, for intrusion detection in AODV-based MANETs. The approach consists of three phases: training, detection, and updating. In the training phase, a niching artificial bee colony algorithm, called *NicheNABC*, runs a negative selection algorithm multiple times to generate a set of mature negative detectors to cover the nonself space. In the detection phase, mature negative detectors are used to discriminate between normal and malicious network activities. In the updating phase, the set of mature negative detectors is updated by one of two methods of partial updating or total updating. We use the Monte Carlo integration to estimate the amount of the nonself space covered by negative detectors and to determine when the total updating should be done. We demonstrate the effectiveness of BeeID for detecting several types of routing attacks on AODV-based MANETs simulated using the NS2 simulator. The experimental results show that BeeID can achieve a better tradeoff between detection rate and false-alarm rate as compared to other dynamic approaches previously reported in the literature.

1 Introduction

Mobile ad hoc networks (MANETs) are self-organized networks of wireless mobile nodes that communicate with each other without the use of any fixed network infrastructure or centralized administration [1]. Every mobile node runs a common routing protocol such as AODV and acts both as a terminal and a router,

☆ This article is an extended/revised version of an ISCISC'11 paper.

* Corresponding author.

Email addresses: fatem.baranibaravati@modares.ac.ir (F. Barani), abadi@modares.ac.ir (M. Abadi).

ISSN: 2008-2045

forwarding packets from one node to another. *Ad hoc on-demand distance vector* (AODV) [2] is a reactive routing protocol that tries to minimize the requirements of broadcast during a route discovery process.

MANETs can be described as open networks with highly dynamic and constantly changing topology. Any mobile node with the proper hardware and enough knowledge of routing protocols is able to connect to them. These networks are suitable for applications in which no fixed network infrastructure exists, such as military battlefield, emergency rescue, and vehicular communications [3].

Due to inherent characteristics of MANETs, It is difficult to guarantee the correct execution of a routing protocol by all nodes. A malicious node can easily listen to the network traffic and launch attacks on other nodes with the purpose of disrupting the normal operation of the network or stealing the information. Many different types of attacks against these networks have been identified including *Flooding*, *Blackhole*, *Neighbor*, *Rushing*, and *Wormhole* attacks [4].

The *intrusion detection* (ID) techniques proposed for MANETs can be grouped into two classes: *signature-based detection* and *anomaly detection*. A signature-based detection technique compares current network behavior with known attack signatures and detects an attack if there is a match. An anomaly detection technique builds a model of normal network behavior and then considers any deviation from this model as anomaly. The advantage of anomaly detection techniques is that they do not require known attack signatures and can thus detect new attacks [5].

In this paper, we use the *negative selection* mechanism of biological immune system for intrusion detection in AODV-based MANETs. It is based on the principles of self/nonself discrimination [6, 7] in the biological immune system, and models the maturation process of T cells in thymus without the participation of nonself cells and eliminates T cells that react against self cells [8]. In other words, it uses only self cells for learning and generates a set of detectors for detecting nonself cells [9]. Previous works have used different techniques for generating detectors [10–13]. In this work, we extend the *artificial bee colony* (ABC) algorithm [14] for this purpose. It is a new swarm intelligence algorithm inspired by the behavior of honey bees when searching for food sources. Many extensions have been proposed to improve its performance [15, 16].

Mobility of nodes in a MANET causes the network topology to change constantly over time. Hence, a malicious node can easily disrupt the routing process by injecting false routes into the network. A traditional approach to detect such malicious network activities is to build a profile of the normal network traffic and then identify an activity as suspicious if it deviates from this profile. As the network topology dynamically changes over time, the simple use of a static profile is not efficient.

In this paper, we present a dynamic hybrid approach based on the ABC and NS algorithms, called *BeeID*, for intrusion detection in AODV-based MANETs. In this approach, every node first collects a set of feature vectors of its own normal network traffic. Each feature vector is represented by a hypersphere with fixed radius in the feature space. The node then applies the NicheNABC algorithm to generate a set of mature negative detectors to cover the nonself space. The amount of the coverage is estimated using the Monte Carlo integration, a probabilistic and sampling method useful for estimating complex integrals. The negative detectors, represented by hyperspheres with variable radii, are used to detect malicious network activities. The node eventually updates mature negative detectors by one of two methods of partial updating or total updating.

The remainder of this paper is organized as follows: Section 2 reviews related works. Section 3 provides a brief overview of ABC and AODV. Section 4 formally introduces the problem of intrusion detection in MANETs and Section 5 presents BeeID. Section 6 reports experimental results and finally Section 7 draws some conclusions.

2 Related Works

Gonzalez *et al.* [17] proposed a *real-valued negative selection* (RNS) algorithm with constant-sized detectors and used the Monte Carlo integration to calculate the number of detectors needed to cover the nonself space. Simulated annealing was employed to optimize the distribution of detectors in the nonself space.

Ji *et al.* [11] presented a real-valued negative selection algorithm with variable-sized detectors, called *V-detector*. A naive method was used to automatically calculate the estimated coverage of the nonself space when the detector set is generated. They further improved the coverage estimation by the hypothesis testing, and demonstrated that it has a higher detection rate than the naive method, but generates more detectors to cover the nonself space.

The security techniques proposed for MANETs can be grouped into two classes: *prevention* and *detection*. Prevention techniques, such as secure and authenticated routing protocols [18, 19], are usually considered as the first line of defense against attacks. However, these techniques do not provide a complete solution

for all attacks. Detection techniques can come into play when prevention techniques have failed.

Dasgupta *et al.* [12] proposed a technique inspired by the negative selection algorithm for intrusion detection in wired networks. It uses a niching genetic algorithm (NGA) to generate a set of detectors to cover the nonself space. The detectors are represented in the form of rules. The condition part of each rule defines a hyperrectangle in the feature space. A hypersphere is defined around each self sample. The raw fitness of a rule is calculated based on the volume of its hyperrectangle and the number of self samples covered by it. Ostaszewski *et al.* [20] presented another technique for network intrusion detection in which both detectors and self samples are represented by hyperrectangles.

Balachandran *et al.* [21] proposed a behavior-based anomaly detection technique inspired by the biological immune system to detect malicious nodes in DSR-based MANETs. They generate the detector set through a *structured genetic algorithm* (sGA) that is suitable for encoding of multi-shape detectors. Sarafijanovic *et al.* [22] used an artificial immune system based on negative selection, danger theory, and clonal selection for detecting malicious nodes.

In some real-valued negative selection algorithms, the variability of self samples would result in the holes on the boundary between the self and nonself spaces. Hence, nonself samples in these regions cannot be detected. Wang *et al.* [13] proposed an improved detector generation algorithm based on evolutionary search to generate a specific type of detectors, called *boundary detectors*. These detectors cover the holes on the boundary and have an opportunity to detect nonself samples hidden in the self space.

Hang *et al.* [23] presented an approach of applying both positive and negative selection algorithms for anomaly detection. They consider the problem of anomaly detection as a problem of supervised learning from imbalanced data sets and use re-sampling strategies to balance data sets. The approach first learns the patterns of normal samples based on a coevolutionary genetic algorithm, which is inspired from the positive selection algorithm, and then generates synthetic anomalous samples based on the negative selection algorithm. Both data sets are used for learning a classifier. The main limitation of this approach is that it imposes a significant overhead for updating the boundary between normal and anomalous samples and therefore is not appropriate for dynamic anomaly detection.

Nakayama *et al.* [24] proposed a dynamic anomaly detection approach, called *WPCA*, for AODV-based MANETs that allows the profile of normal network behavior to be updated at particular time intervals. It uses the *principal component analysis* (PCA) to calculate the first principle component of normal samples, which can be used as a profile of normal network behavior. The projection distance of new samples to this principle component is used for detecting routing attacks. The global covariance of normal samples is used to update the profile at consecutive time intervals. The main drawback of this approach is that the global covariance is calculated inaccurately.

Alikhany *et al.* [5] proposed a dynamic clustering-based approach, called *DCAD*, for anomaly detection in AODV-based MANETs. It uses a weighted fixed-width clustering algorithm to build a profile of normal network behavior and to detect routing attacks. It also uses a forgetting equation to periodically update the profile. The experimental results have shown that DCAD has a high false-alarm rate.

3 Background Knowledge

3.1 Artificial Bee Colony

The *artificial bee colony* (ABC) algorithm [14] is a new swarm intelligence algorithm inspired by the behavior of honey bees. Since it was proposed by Karaboga [25], many extensions have been made to improve it [15, 16]. Karaboga *et al.* [16] compared the performance of ABC with that of some other popular metaheuristic optimization algorithms, such as *particle swarm optimization* (PSO), *genetic algorithms* (GAs), and *differential evolution* (DE). The results showed that it has a comparable performance with other algorithms.

The colony of artificial bees consists of three groups of bees: *employed*, *onlooker*, and *scout*. The number of employed and onlooker bees is identical, and is equal to the number of food sources. Each employed bee is assigned to one of food sources, and in each cycle it finds a new food source in the neighborhood of its current food source. If the new food source has more nectar, the employed bee will replace the current food source with it. There is only one scout bee in the colony. The employed bee whose food source has been abandoned becomes a scout bee, and carries out a random search to find a new food source [14]. After all employed bees complete the search process, they share the information about the nectar amount and the position of food sources with onlooker bees by doing the waggle dance. Each onlooker bee watches the dance and selects a food source based on its nectar amount. The position of each food source represents a possible solution in the optimization problem and its nectar amount corresponds to the fitness of the solution [16].

3.2 An Overview of AODV Protocol

Different routing protocols for MANETs have been proposed in the literature, which can be classified into two types: *table driven* and *on-demand routing* protocols [26]. In on-demand routing protocols, routes are created only when required. Some of these protocols are DSR [27], AODV [2], and TORA [28].

AODV [2], which is based on DSDV [26] and DSR, tries to minimize the requirements of broadcast during a route discovery process when a source node wants to send data to a destination node. The source node broadcasts a route request (RREQ) packet to its neighbors and then sets a timer to wait for a reply. A route reply (RREP) packet is sent back to the source node by the destination node or any intermediate node that has a fresh route to the destination node. Every node in the route between the source and destination nodes processes the RREQ packet to create a reverse route in its routing table for forwarding RREP packets to the source node [5].

3.3 Typical Routing Attacks on AODV Protocol

Routing attacks on AODV are classified into four classes: (1) Route disruption, (2) Route invasion, (3) Node isolation, and (4) Resource consumption. In the following, we shortly describe some of typical routing attacks on AODV [4, 24].

1. *Flooding Attack*: A malicious node sends a huge number of RREQ packets in an attempt to consume the network resources. The source IP address is forged to a randomly selected node and the broadcast ID is intentionally increased.
2. *Blackhole Attack*: A malicious node advertises itself as having a valid route to a destination node. For this purpose, after receiving a RREQ packet via broadcast, it sends a forged RREP packet back to the source node with a greater sequence number. Therefore, the source node imagines that the malicious node has a fresh route to the destination node and drops other received RREP packets. The malicious node takes all the routes towards itself and does not allow forwarding any packet anywhere (see Figure 1).
3. *Neighbor Attack*: A malicious node forwards RREQ/RREP packets without adding its ID in them. This causes two nodes that are not within the communication range of each other imagine that they are neighbors, resulting in a disrupted route.
4. *Rushing Attack*: On-demand routing protocols prevent the collisions of RREQ packets by using a delay between receiving a RREQ packet and forwarding it. A malicious node exploits this property of these routing protocols by quickly forwarding RREQ packets. Hence, the source node cannot discover any valid route that does not contain the malicious node.
5. *Wormhole Attack*: Two or more malicious colluding nodes first establish a private high-speed link, referred to as *Wormhole link*, between themselves in the network. They then record RREQ packets at one location, tunnel them to other location through the Wormhole link without increasing the value of the hop count field, and then rebroadcast them into the network [29]. This attack can cause serious damages to the network and prevent the discovery of any route other than through the Wormhole link [4] (see Figure 2).

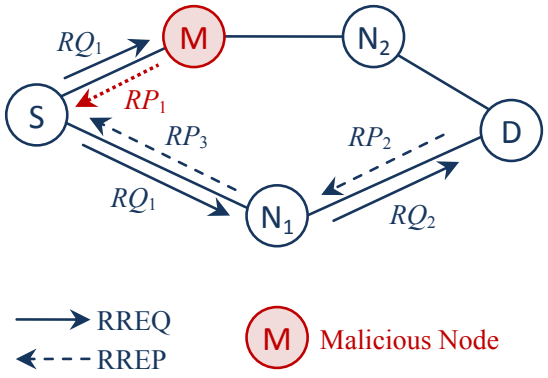


Figure 1. A Blackhole attack with forged RREP packets

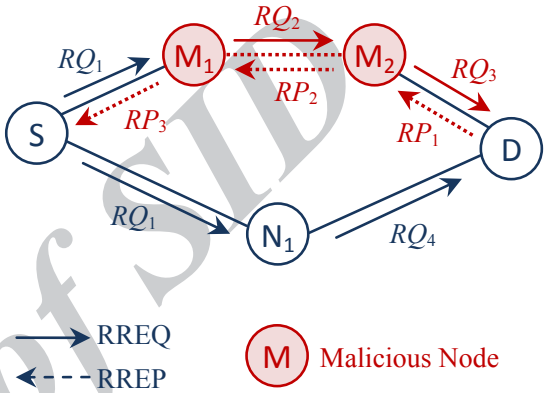


Figure 2. A Wormhole attack with two malicious nodes

4 Problem Definition

A mobile ad hoc network (MANET) is a collection of wireless mobile nodes that are capable of communicating with each other without the use of any fixed network infrastructure or centralized management. During each time slot Δt_i, every mobile node collects statistics from its own network traffic and represents them as a $p-$dimensional feature vector x_i:

$$x_i = (x_i^1, x_i^2, ..., x_i^p), \quad (1)$$

where each $x_i^k \in [0.0, 1.0]$ is a measurable feature. A set of contiguous time slots is referred to as a *time window*.

The set of all possible feature vectors constitutes the feature space $S \subseteq [0.0, 1.0]^p$, where a feature vector $x_i \in S$ is associated with an antigen in the biological immune system and collected from the network traffic by the *antigen presenting cells* (APCs). A feature vector $x_i \in S$ in a time-window t is classified as normal if it corresponds to the normal network state in this time-window. The set of all normal feature vectors in the current time-window t is denoted as $X_{\text{self}}(t)$, and the set of all normal feature vectors in the last m time-windows is denoted as $N_{\text{self}}(t)$:

$$N_{\text{self}}(t) = \bigcup_{\tau=t-m+1}^{t} X_{\text{self}}(\tau). \quad (2)$$

The normal feature vectors in $N_{\text{self}}(t)$ are covered by a set $PA(t)$ of positive antigens. A positive antigen $p_i \in PA(t)$ is defined as a hypersphere $p_i = (x_i, r_{\text{self}})$ centered at the normal feature vector $x_i \in N_{\text{self}}(t)$ with a constant radius r_{self}.

$$PA(t) = \{(x_i, r_{\text{self}}) | x_i \in N_{\text{self}}(t)\}. \quad (3)$$

Our aim is to generate a set $ND(t)$ of negative detectors that have the maximum coverage of the nonself space $S - PA(t)$ in the time-window t. Every negative detector $d_j \in ND(t)$ is defined as a hypersphere $d_j = (c_j, r_j)$, where $c_j = (c_j^1, c_j^2, \ldots, c_j^p)$ is the center of the hypersphere in the nonself space $S - PA(t)$ and r_j is its radius.

A feature vector $x_i \in S$ in the time-window t is classified as malicious if and only if it is covered by at least a negative detector $d_j \in ND(t-1)$. Otherwise, it is classified as normal:

$$\begin{cases} cv(x_i, d_j) = \text{true for some } d_j \in ND(t-1) & : \text{Malicious}, \\ cv(x_i, d_j) = \text{false for all } d_j \in ND(t-1) & : \text{Normal}, \end{cases} \quad (4)$$

where $cv(x_i, d_j)$ denotes whether the feature vector x_i is covered by the negative detector d_j:

$$cv(x_i, d_j) = \begin{cases} \text{true} & \text{if } dis(x_i, c_j) \le r_j, \\ \text{false} & \text{otherwise}, \end{cases} \quad (5)$$

where $dis(x_i, c_j)$ is the Euclidean distance between feature vectors x_i and c_j. The positive antigen $p_i \in PA(t)$ overlaps with the negative detector d_j if their intersection in the feature space is not empty:

$$ovlp(p_i, d_j) = \begin{cases} \text{true} & \text{if } dis(x_i, c_j) \le (r_{\text{self}} + r_j), \\ \text{false} & \text{otherwise}. \end{cases} \quad (6)$$

5 BeeID Approach

In this section, we present a dynamic hybrid approach based on the ABC and NS algorithms, called *BeeID*, for intrusion detection in MANETs. The approach consists of three phases: *training*, *detection*, and *updating*. In the training phase, negative detectors are generated in a protected environment using the normal network traffic; while in the updating phase, those are generated in a non-protected environment, where routing attacks may occur.

In the training phase, every node first collects a set $N_{\text{self}}(0)$ of feature vectors of its own normal network traffic and assigns the initial weight w_0 to them. It then normalizes the values of each feature, and applies the NicheNABC algorithm on the set $PA(0)$ of positive antigens to generate a set $ND(0)$ of mature negative detectors.

In the detection phase, during each time-window t, every node collects a set $X(t)$ of feature vectors of its own network traffic, and normalizes the values of each feature based on the minimum and maximum values of that feature in $N_{\text{self}}(t-1)$. It then compares the feature vectors with mature negative detectors in $ND(t-1)$ to detect malicious network activities.

In the updating phase, at the end of the time-window t, every node adds the set $X_{\text{self}}(t)$ of normal feature vectors to $N_{\text{self}}(t-1)$:

$$N_{\text{self}}(t) = N_{\text{self}}(t-1) \cup X_{\text{self}}(t). \quad (7)$$

It then updates the weight of each feature vector in $N_{\text{self}}(t)$ using a forgetting equation [5]:

$$w_\tau(t) = \begin{cases} w_0 e^{-r(\tau,t)\Delta T(t-\tau)} & (t-m) < \tau \le t, \\ 0 & \text{otherwise}, \end{cases} \quad (8)$$

where w_0 is a parameter specified by the user. $w_\tau(t)$ is the current weight assigned to all normal feature vectors collected in the time-window τ ($\tau \le t$). $r(\tau, t)$ is the rate of changing of network topology [5] between time-windows τ and t. ΔT is the length of a time-window. A feature vector is eliminated from $N_{\text{self}}(t)$ if its weight is less than a given threshold. Based on the above equation, the weight of normal feature vectors is decreased over each time-window. In other words, the older the normal feature vector, the lower the assigned weight.

It eventually updates the set $ND(t-1)$ of mature negative detectors by one of two updating methods: *partial updating* and *total updating*.

5.1 Normalization Process

When calculating the distance between two feature vectors, features with larger values will dominate those with smaller values, and hence have a larger influence on the calculated distance. To solve this problem, at the end of each time-window t, we normalize the values of each feature in $N_{\text{self}}(t)$.

As mentioned before, the feature space S is a unit hypercube. To generate an optimized set of negative detectors, the values of features in $N_{\text{self}}(t)$ should be normalized such that positive antigens in $PA(t)$ are away from the sides of the unit hypercube S. For this purpose, the minimum and maximum values of each feature in $N_{\text{self}}(t)$ are determined and then the value of the k^{th} feature for each feature vector $x_i \in N_{\text{self}}(t)$ is scaled using the following equation:

$$\hat{x}_i^k = \frac{x_i^k - (x_{\min}^k - 0.2 * x_{\min}^k)}{(x_{\max}^k + 0.2 * x_{\max}^k) - (x_{\min}^k - 0.2 * x_{\min}^k)}, \tag{9}$$

where $x_{\min}^k$ and $x_{\max}^k$ are the minimum and maximum values of the k^{th} feature.

5.2 NicheNABC Algorithm

The NicheNABC algorithm is composed of a number of sequential and dependent niches of the NABC algorithm. The operation of the algorithm is similar to that of the bone marrow and thymus in the biological immune system.

Let t be the current time-window and $PA(t)$ be the set of positive antigens corresponding to the normal feature vectors in $N_{\text{self}}(t)$. The NicheNABC algorithm takes $PA(t)$ as input and generates a set $ND(t)$ of mature negative detectors as output to cover the nonself space $S - PA(t)$. In every niche i of the NABC algorithm, an initial population $F^i = \{f_1^i, f_2^i, \ldots, f_L^i\}$ of immature food sources is randomly generated. Each immature food source $f_j^i \in F^i$ consists of l immature negative detectors d_{js}^i:

$$f_j^i = (d_{j1}^i, d_{j2}^i, \ldots, d_{jl}^i), \tag{10}$$

where d_{js}^i is defined as a hypersphere $d_{js}^i = (c_{js}^i, r_{js}^i)$ whose center is a p-dimensional feature vector $c_{js}^i \in S - PA(t)$ and radius is r_{js}^i, calculated as the average distance between c_{js}^i and its n nearest positive antigens in $PA(t)$. The position and size of f_j^i are denoted by $p_j^i = (c_{j1}^i, c_{j2}^i, \ldots, c_{jl}^i)$ and $s_j^i = (r_{j1}^i, r_{j2}^i, \ldots, r_{jl}^i)$, respectively.

The immature food sources are generated so that they have minimum overlap with positive antigens in $PA(t)$. Hence, each center $c_{js}^i \in p_j^i$ is initially chosen randomly from $S - PA(t)$.

From Section 3.1, we know that each employed bee is assigned to only one of the food sources. In each cycle of niche i, every employed bee j first produces a modification on the position p_j^i of its own food source f_j^i in its memory to discover a new food source $\hat{f}_j^i$, and evaluates the nectar amount of $\hat{f}_j^i$. To do this, it modifies the center c_{js}^i of each negative detector $d_{js}^i \in f_j^i$ with probability P_e:

$$\hat{c}_{js}^i = \begin{cases} c_{js}^i + \varphi_{js}(c_{js}^i - c_{ws}^i) & \text{if } U(0,1) \le P_e, \\ c_{js}^i & \text{otherwise,} \end{cases} \tag{11}$$

where c_{ws}^i is the center of negative detector d_{ws}^i of a randomly chosen food source f_w^i and φ_{js} is a random value in the range $[-1, 1]$.

The employed bee j then forgets f_j^i and replaces it with $\hat{f}_j^i$, only if the nectar amount of $\hat{f}_j^i$ is equal or more than that of f_j^i. In other words, a greedy strategy is applied to select between f_j^i and $\hat{f}_j^i$. The nectar amount of each food source is evaluated by the fitness function ϑ, as in (13).

After all employed bees complete the search process, they share the information of the food sources in F^i with the onlooker bees on the dance area.

Each onlooker bee o first chooses a food source f_k^i using a selection method such as roulette wheel or tournament selection, and then discovers a new food source $\hat{f}_k^i$ in the neighborhood of f_k^i. To this end, it modifies the center c_{ks}^i of each negative detector $d_{ks}^i \in f_k^i$ with probability P_o:

$$\hat{c}_{ks}^i = \begin{cases} c_{ks}^i + \varphi_{ks}(c_{ks}^i - c_{zs}^i) & \text{if } U(0,1) \le P_o, \\ c_{ks}^i & \text{otherwise,} \end{cases} \tag{12}$$

where c_{zs}^i is the center of negative detector d_{zs}^i of a randomly chosen food source f_z^i, and φ_{ks} is a random value in the range $[-1, 1]$. It eventually applies a greedy strategy to select between f_k^i and $\hat{f}_k^i$, similar to that of employed bees.

If the fitness of a food source cannot be improved further for a given number of cycles, that food source is determined to be abandoned. Hence, the scout bee chooses the abandoned food source with the lowest nectar amount and replaces it with a randomly generated immature food source.

The above steps are repeated until a termination condition (e.g., the maximum cycle number) is met. At the end of every niche i, mature negative detectors of the food source f_M^i with the highest nectar amount are added to $ND(t)$. If i is a multiple of ξ, the coverage of the nonself space by mature negative detectors in $ND(t)$ is estimated using the Monte Carlo integration [30, 31]. The sequential niches of the NABC algorithm

are continued until a termination condition (e.g., the maximum niche number or the target coverage of the nonself space) is met. Algorithm 1 shows the pseudo-code of NicheNABC. In the algorithm, $R_{\max}$ is the maximum number of niches, cv_{est} is the estimated coverage of the nonself space, and $CR_{\min}$ is the target coverage of the nonself space.

Algorithm 1 NicheNABC

Input:
$PA(t)$: Set of positive antigens
$R_{\max}$: Maximum number of niches
$CR_{\min}$: Target coverage of the nonself space
Output:
$ND(t)$: Set of mature negative detectors

$i := 1$, $ND(t) := \phi$
repeat
  Initialize a population of immature food sources
  repeat
    for each employed bee j **do**
      Produce a new food source $\hat{f}_j^i$ in the neighborhood of f_j^i using Equation 11
      if $\vartheta(\hat{f}_j^i, PA(t), ND(t)) > \vartheta(f_j^i, PA(t), ND(t))$ **then**
        $f_j^i := \hat{f}_j^i$
      end if
    end for
    for each onlooker bee o **do**
      Select a food source f_k^i using tournament selection and product a new food source $\hat{f}_k^i$ in the neighborhood of f_k^i using Equation 12
      if $\vartheta(\hat{f}_k^i, PA(t), ND(t)) > \vartheta(f_k^i, PA(t), ND(t))$ **then**
        $f_k^i := \hat{f}_k^i$
      end if
    end for
    Determine an abandoned food source and replace it with a new immature food source for the scout bee
    Find the best food source f_M^i
  until a termination condition is met
  $ND(t) := ND(t) \cup f_M^i$
  $i := i + 1$
  if i is a multiple of ξ **then**
    Calculate the nonself space coverage estimation cv_{est} using the Monte Carlo integration
  end if
until $i > R_{\max}$ or $cv_{\text{est}} > CR_{\min}$

5.2.1 Fitness Function

The NicheNABC algorithm attempts to generate a set of mature food sources with high nectar amount, so that they have the maximum coverage with the nonself space, the minimum overlap with positive antigens, and the minimum overlap among themselves.

In every niche of the NABC algorithm, the nectar amount of a food source f is evaluated by the following fitness function:

$$\begin{aligned}\vartheta(f, A, D) = w_1 \cdot vol(f) - w_2 \cdot ovlp_{\text{self}}(f, A) \\ -w_3 \cdot ovlp_{\text{inner}}(f) - w_4 \cdot ovlp_{\text{outer}}(f, D),\end{aligned} \quad (13)$$

where A is the set of positive antigens, D is the set of mature negative detectors generated in the previous niches of the algorithm, $w_1, \ldots, w_4$ are constants whose values are determined experimentally, and $vol(f)$ denotes the approximate amount of the feature space S covered by negative detectors in f:

$$vol(f) = \sum_{s=1}^{l} \frac{r_s}{\sqrt{p}}, \quad (14)$$

where l is the number of negative detectors in a food source, p is the dimension of S, and r_s is the radius of s^{th} negative detector in f. The larger the radius, the larger the amount of S covered by the negative detector. Hence, the sum of the radii of negative detectors in f is used to estimate the amount of S covered by it.

$ovlp_{\text{self}}(f, A)$ denotes the approximate amount of overlap between negative detectors in f and positive antigens in A:

$$ovlp_{\text{self}}(f, A) = \frac{\sum_{s=1}^{l} \sum_{k=1}^{|A|} (e^{\delta(d_s, p_k)} - 1)}{l \cdot |A|}, \quad (15)$$

where d_s is the s^{th} negative detector in f, p_k is the k^{th} positive antigen in A, and $|A|$ is the number of positive antigens in it.

$ovlp_{\text{inner}}(f)$ denotes the approximate amount of overlap among negative detectors in f:

$$ovlp_{\text{inner}}(f) = \frac{\sum_{s=1}^{l} \sum_{k=s+1}^{l} (e^{\delta(d_s, p_k)} - 1)}{l \cdot (l+1)}. \quad (16)$$

$ovlp_{\text{outer}}(f, D)$ denotes the approximate amount of overlap between negative detectors in f and mature negative detectors in D:

$$ovlp_{\text{outer}}(f, D) = \begin{cases} \frac{\sum_{s=1}^{l} \sum_{k=1}^{|D|} (e^{\delta(d_s, m_k)} - 1)}{l \cdot |D|} & \text{if } D \neq \varnothing, \\ 0 & \text{otherwise,} \end{cases} \quad (17)$$

where m_k is the k^{th} mature negative detector in D, and $|D|$ is the number of mature negative detectors in it.

Notice that $\delta(s_j, s_k)$ denotes the approximate amount of overlap between two hyperspheres $s_j = (c_j, r_j)$ and $s_k = (c_k, r_k)$, and its value is always bounded between 0 and 1:

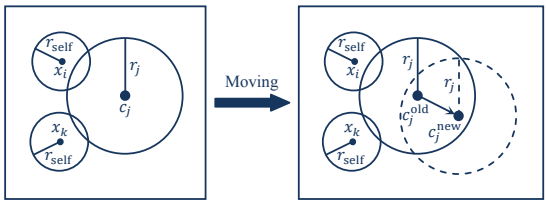


Figure 3. The partial updating of negative detectors in the updating phase

$$\delta(s_j, s_k) = 1 - \frac{dis(c_j, c_k)}{r_j + r_k}, \tag{18}$$

where $dis(c_j, c_k)$ is the Euclidean distance between two centers c_j and c_k of hyperspheres s_j and s_k, respectively. The maximum value of δ is reached when $dis(c_j, c_k) = 0$ and its minimum value is reached when $dis(c_j, c_k) \geqslant r_j + r_k$.

5.3 Partial Updating

Let t be the current time-window and $A_X(t)$ be the set of positive antigens corresponding to the normal feature vectors in $X_{\text{self}}(t)$:

$$A_X(t) = \{(x_i, r_{\text{self}}) | x_i \in X_{\text{self}}(t)\}. \tag{19}$$

If t is not a multiple of a given integer parameter ψ, mature negative detectors in $ND(t-1)$ are updated using the partial updating process.

If a mature negative detector $d_j \in ND(t-1)$ overlaps with at least one of positive antigens in $A_X(t)$, its center is moved away from its nearest positive antigen, but its radius remains unchanged [10]:

$$\begin{cases} c_j^{\text{new}} = c_j^{\text{old}} + \alpha \cdot \dfrac{dir}{\|dir\|}, \\ dir = c_j^{\text{old}} - x_i, \end{cases} \tag{20}$$

where c_j^{old} and c_j^{new} are the center of d_j before and after moving it, respectively, x_i is the center of the nearest positive antigen to d_j, α is a predefined parameter that controls the amount of movement, and $\|\cdot\|$ denotes the Euclidean norm. Figure 3 shows the partial updating process.

5.4 Total Updating

Let t be the current time-window. If t is a multiple of a parameter ψ, the coverage of the nonself space by negative detectors in $ND(t-1)$, denoted as cv_{est}, is estimated using the Monte Carlo integration. If cv_{est} is less than a given threshold $CU_{\min}$, the NicheNABC algorithm is applied on the set $PA(t)$ of positive antigens to generate a set $ND(t)$ of new mature negative detectors. This process is called *total updating*. Algorithm 2 shows the pseudo code of the detection and updating phases.

Algorithm 2 Detection and Updating

Input:
$ND(0)$: Set of mature negative detectors
$N_{self}(0)$: Set of normal feature vectors
$CU_{\min}$: Minimum coverage of the nonself space
ψ: Period of the total updating

$t := 1$
while true **do**
  $X_{\text{self}}(t) := \phi$
  Collect the set of feature vectors $X(t)$ from the network traffic
  Normalize the values of feature vectors in $X(t)$ using Equation 9
  for each feature vector $x_i \in X(t)$ **do**
    if $cv(x_i, d_j)$ is true for some $d_j \in ND(t-1)$ **then**
      Classify x_i as malicious
    else
      $X_{\text{self}}(t) := X_{\text{self}}(t) \cup \{x_i\}$
    end if
  end for
  $N_{\text{self}}(t) := N_{\text{self}}(t-1) \cup X_{\text{self}}(t)$
  Update the weights of feature vectors in $N_{\text{self}}(t)$
  if t is a multiple of ψ **then**
    Calculate the nonself space coverage estimation cv_{est} using the Monte Carlo integration
    if $cv_{\text{est}} < CU_{\min}$ **then**
      Generate $ND(t)$ using the NicheNABC algorithm
    end if
  else
    $A_X(t) = \{(x_i, r_{\text{self}}) | x_i \in X_{\text{self}}(t)\}$
    for each negative detector $d_j \in ND(t-1)$ **do**
      if $ovlp(p_i, d_j)$ is true for some $p_i \in A_X(t)$ **then**
        Move d_j using Equation 20
      end if
    end for
    $ND(t) := ND(t-1)$
  end if
  $t := t + 1$
end while

6 Experiments

In order to evaluate the performance of BeeID, we conducted some experiments using the data collected from a series of simulations. We used two performance measures: *detection rate* (DR) and *false-alarm rate* (FAR). The experimental results are presented in this section.

Table 1. Simulation parameters in NS2

Parameter Name	Parameter Value
Simulation Time	10, 000(s)
Number of Mobile Nodes	30
Simulation Area	1000(m)×1000(m)
MAC Layer Protocol	MAC 802.11
Routing Protocol	AODV
Traffic Model	CBR
Pause Time	5(s)
Mobility Model	RWP
Maximum Mobility	35(m/s)
Maximum Bandwidth	2(Mbps)
Packet Size	512(B)

6.1 Simulation Environment

The NS2 simulator version 2.29 [32] was used to simulate some routing attacks on MANETs. It provides an excellent environment to simulate various wireless networks. The simulation was performed for a 30-node network with a square topology of dimensions 1000(m)∗1000(m). IEEE 802.11 and AODV were used as the MAC layer and routing protocols, respectively. The simulation time was set to 10, 000(s). One certain node was selected to launch Flooding, Blackhole, Neighbor, and Rushing attacks, and two certain nodes were selected to launch a Wormhole attack. The *constant bit rate* (CBR) traffic with a packet size of 512 bytes was generated by running the *cbrgen.tcl* program. The *random waypoint* (RWP) model was used as the mobility model and generated by the *setdest* program. It is a commonly used synthetic model for node mobility in MANETs in which every node independently and randomly chooses its destination and speed. Table 1 summarizes the simulation parameters. All the experiments were done on a computer with a 1.8 GHz Intel Core 2 Duo processor and 512 MB RAM.

At each time slot, every mobile node collected a feature vector of its own network traffic. The length of each time slot was set to 5(s). We used 22 features, presented in [5, 24]. As can be seen in Table 2, the features are classified into four categories:

- Three CBR traffic features (CBR),
- Ten route discovery features (RTD),
- Five path disrupting features (PTD),
- Four protocol specific features (PRT).

Table 2. List of features

Type	#	Feature
CBR	1	Number of sent CBR data packets
	2	Number of received CBR data packets
	3	Number of forwarded CBR data packets
RTD	4	Number of sent RREQ packets
	5	Number of received RREQ packets with the same source address as the node
	6	Number of received RREQ packets with the same destination address as the node
	7	Number of received RREQ packets with the different source and destination address of the node
	8	Number of forwarded RREQ packets
	9	Number of sent RREP packets with the same destination address as the node
	10	Number of sent RREP packets with the different destination address of the node
	11	Number of received RREP packets with the same source address as the node
	12	Number of received RREP packets with the different source address of the node
	13	Number of forwarded RREP packets
PTD	14	Number of sent RERR packets
	15	Number of received RERR packets
	16	Number of forwarded RERR packets
	17	Number of dropped RREQ packets
	18	Number of dropped RREP packets
PRT	19	Average difference between the destination sequence number of each received RREP packet and its corresponding sequence number stored in the routing table
	20	Maximum difference between the hop count of each received RREQ packet and its corresponding hop count stored in the routing table
	21	Average difference between the hop count of each received RREP packet and its corresponding hop count stored in the routing table
	22	Average sequence numbers of active entries in the routing table

6.2 Results

The length of the training phase was set to 1500(s). During this phase, we prohibited nodes from performing any malicious activities. Every node collected a set of feature vectors (positive antigens) of its own normal network traffic and generated a set of mature negative detectors. During the detection phase, it used the mature negative detectors to discriminate

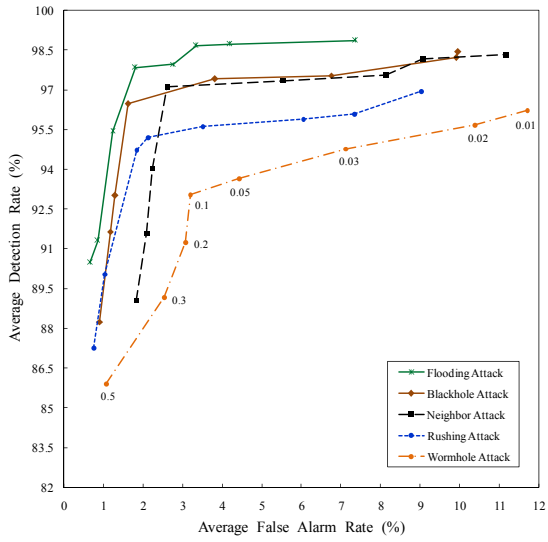


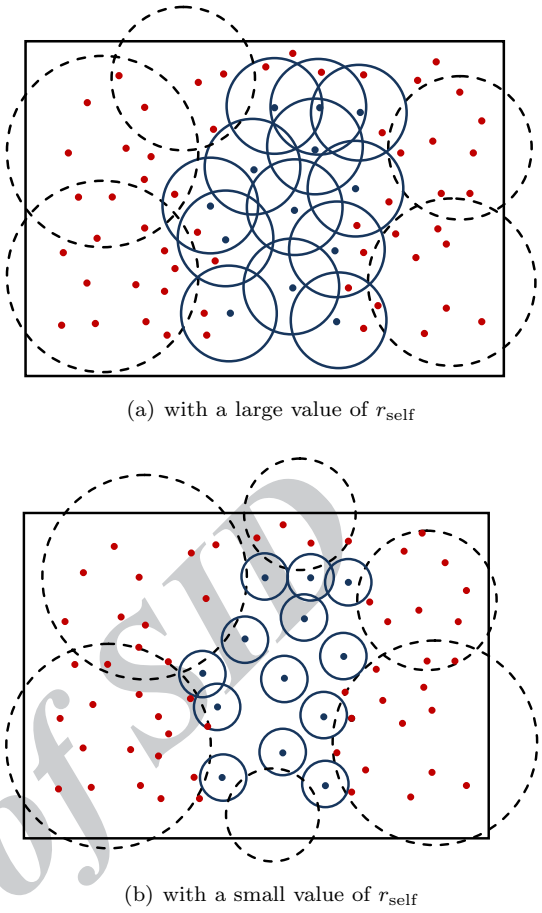
Figure 4. Performance of BeeID for different values of r_{self}

between normal and malicious network activities. During the updating phase, the node updated the mature negative detectors by one of two methods of partial updating or total updating. Through experiments, the parameters m and ψ were set to 10. The parameter r_{self} was set to 0.1. In addition, the number of the nearest positive antigens used to calculate the radius of a negative detector was set to $n = 3$, and the length of the time-window was set to $\Delta T = 100$(s).

The malicious node(s) launched five different routing attacks, including Flooding, Blackhole, Neighbor, Rushing, and Wormhole from 3500(s) to 6000(s). Every node, except the malicious node(s), used its own generated mature negative detectors to detect malicious network activities.

Figure 4 compares the performance of BeeID to detect the routing attacks for different values of r_{self}, ranging from 0.5 to 0.01. Based on the results shown in the figure, $r_{\text{self}} = 0.1$ is a better choice than other values, because BeeID can better balance between the detection and false-alarm rates. As mentioned before, r_{self} is responsible for defining the boundaries of space covered by positive antigens. An increase in the value of r_{self} leads to an increase in the amount of space covered by positive antigens and a decrease in the amount of the nonself space covered by negative detectors. Hence, the generated mature negative detectors will not be able to cover all possible malicious network activities and therefore the detection rate will decrease (See Figure 5).

Figure 6 depicts the average detection and false-alarm rates of BeeID for different values of ΔT, ranging from 500(s) to 40(s). As shown in the figure, the lower the length of time-window is, the higher the detection and false-alarm rates are. This is because with



(a) with a large value of r_{self}

(b) with a small value of r_{self}

Figure 5. Performance of BeeID for different values of r_{self} in a 2-dimensional feature space. The blue and red points are normal and malicious feature vectors, respectively. The circles with dashed lines are negative detectors and the other circles are positive antigens

decreasing the length of each time-window, the updating period of negative detectors is shortened, and the number of positive antigens decreases. According to the results shown in the figure, a better choice for ΔT is 100(s).

Figure 7 shows the effect of the different values of n, ranging from 1 to 8, on the performance of BeeID to detect the routing attacks. An increase in the value of n causes an increase in both detection and false-alarm rates, because mature negative detectors further violate the self space, but instead they can cover a greater amount of the feature space. Therefore, they will cover more normal and malicious feature vectors. As can be seen in the figure, there is a better balance between the detection rate and false-alarm rate in $n = 3$.

Table 3 shows the effect of different values of ψ on the average detection rate and average run time of the updating phase in each time-window. It should be noted that the lower the value of ψ is, the greater the number of running the total updating and the less

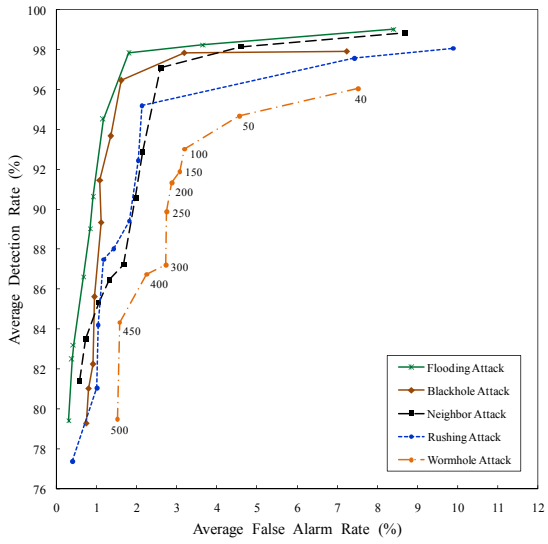


Figure 6. Performance of BeeID for different values of ΔT

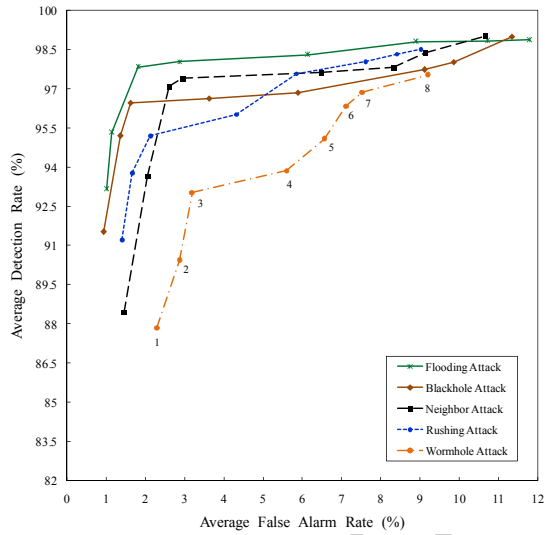


Figure 7. Performance of BeeID for different values of n

the number of running the partial updating in different time-windows. Hence, since the total updating is a time consuming process, the average run time of the updating phase in each time-window will increase. However, it generates mature negative detectors that cover the nonself space better than those of the partial updating. According to the results shown in the table, $\psi = 10$ is a good choice to balance between the detection rate and average run time.

In order to evaluate the effect of the partial updating on the amount of overlap between mature negative detectors and positive antigens, we calculated the average number of collisions between them before and after performing this process. As shown in Figure 8, the partial updating reduces the average number of collisions between mature negative detectors and positive antigens for the routing attacks. On average, there

Table 3. Effect of different values of ψ on the average detection rate and average run time of the updating phase in each time-window

ψ	Average Detection Rate (%)	Average Run Time (ms)
1	96.64	2050.79
2	96.61	1478.61
5	96.04	709.54
10	95.92	368.35
20	94.53	227.58
30	92.12	165.64
40	91.51	152.81
50	89.42	75.67

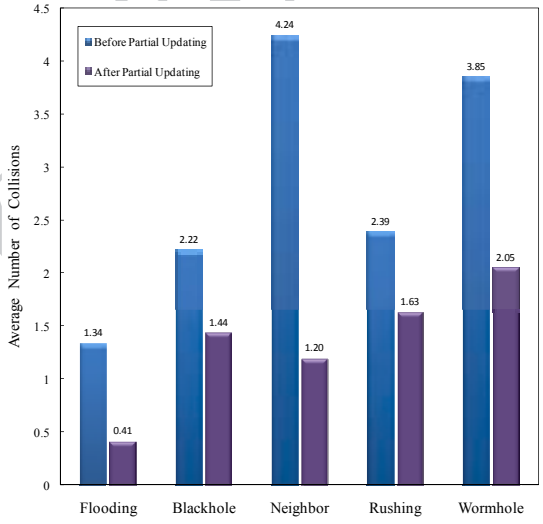


Figure 8. Effect of the partial updating on the amount of overlap between mature negative detectors and positive antigens

is a 69%, 35%, 72%, 32%, and 47% reduction in the number of collisions for Flooding, Blackhole, Neighbor, Rushing, and Wormhole attacks, respectively.

We implemented the NABC algorithm using different selection methods, including roulette wheel selection and binary tournament selection. As shown in Table 4, both of these methods have the same effect on the performance of BeeID. However, binary tournament selection has a lower time complexity than roulette wheel selection. Hence, we used it as the selection operator in the NABC algorithm.

As mentioned before, the NicheNABC algorithm is composed of a number of sequential and dependent niches of the NABC algorithm. Every niche of this algorithm increases the coverage of the nonself space by mature negative detectors. Figure 9 shows the es-

Table 4. Effect of using different selection methods in the NABC algorithm on the performance of BeeID

Attack	Roulette Wheel		Binary Tournament	
	DR	*FAR*	*DR*	*FAR*
Flooding	97.80	2.04	97.83	1.81
Blackhole	96.32	1.77	96.46	1.62
Neighbor	96.81	2.55	97.10	2.60
Rushing	95.23	2.21	95.20	2.13
Wormhole	92.65	3.16	93.03	3.20
Average	95.76	2.35	95.92	2.27

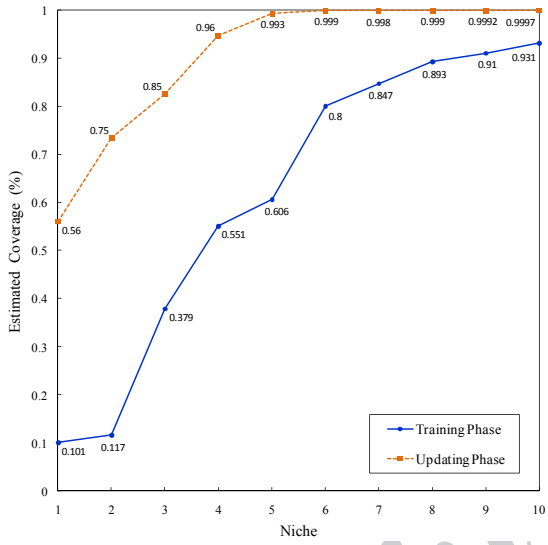


Figure 9. Estimated coverage of the nonself space at every niche of the NABC algorithm for both the training and updating phases

timated coverage of the nonself space at every niche of the NABC algorithm during a complete running of the NicheNABC algorithm for both the training and updating phases. This estimation was calculated using the Monte Carlo integration. The NicheNABC algorithm will be stopped when the coverage of the nonself space is equal or more than a target threshold. As shown in the figure, in the updating phase, the NicheNABC algorithm is able to reach a higher coverage of the nonself space with a less number of niches. This is because the number of positive antigens in the updating phase is less than that of the training phase. Hence, the NABC algorithm in the updating phase will be able to generate larger mature negative detectors to cover the nonself space and faster reach to the target coverage.

During two consecutive running of the total updating, due to the removal of old normal feature vectors and the lack of generating new mature negative detectors, the coverage of the nonself space by existing mature negative detectors gradually decreases. Figure 10 shows the estimated coverage of the nonself space at the end of each time-window.

Table 5. Comparison of the performance of three variants of BeeID

Attack	Last Time Window		Without Updating		With Updating	
	DR	*FAR*	*DR*	*FAR*	*DR*	*FAR*
Flooding	91.80	8.14	82.62	8.81	97.83	1.81
Blackhole	90.06	6.27	86.03	10.73	96.46	1.62
Neighbor	86.12	5.46	78.26	9.46	97.10	2.60
Rushing	86.53	8.83	56.33	15.58	95.20	2.13
Wormhole	81.45	10.30	56.51	14.62	93.03	3.20
Average	87.19	7.80	71.95	11.84	95.92	2.27

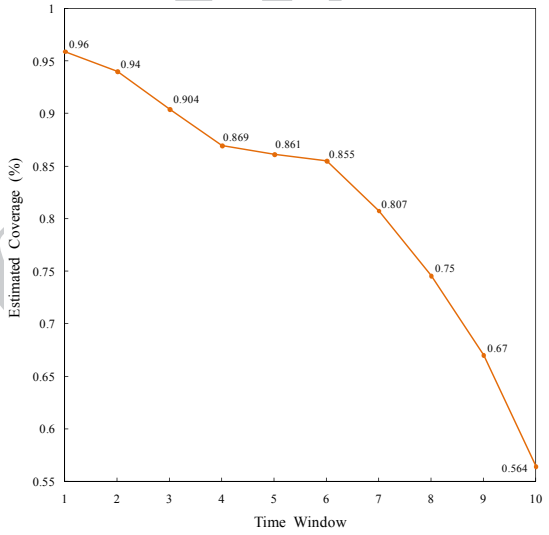


Figure 10. Estimated coverage of the nonself space by mature negative detectors during two consecutive running of the total updating

We evaluated the performance of different variants of BeeID: BeeID with updating, BeeID without updating, and BeeID with last time-window updating. The results can be seen in Table 5. BeeID without updating does not consist of the updating phase. Hence, mature negative detectors are generated in the training phase and are not updated during the next time-windows. BeeID with last time-window updating only uses positive antigens of the last time-window to update mature negative detectors. As can be seen in the table, BeeID with updating has the highest detection rate and the lowest false-alarm rate. It increases the average detection rate by more than 23.97% and decreases

Table 6. Comparison among the average run times of BeeID, DCAD, and WPCA

Attack	BeeID	WPCA	DCAD
Flooding	394.83	207.60	51.70
Blackhole	412.34	235.44	108.04
Neighbor	445.76	208.62	100.78
Rushing	342.34	217.43	55.55
Wormhole	394.53	221.36	103.96
Average	397.96	218.09	84.01

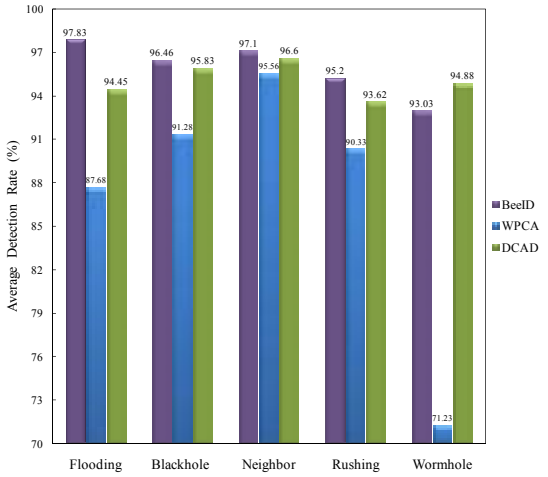
the average false-alarm rate by more than 9.57% of the corresponding values of BeeID without updating. These results illustrate the effect of the partial and total updating on the performance of BeeID.

Figure 11 compares the performance of BeeID with that of DCAD [5] and WPCA [24] for detection of the routing attacks. As shown in the figure, BeeID increases the average detection rate of DCAD and WPCA by more than 0.85% and 8.71% and decreases the average false-alarm rate of them by more than 6.48% and 3.61%, respectively.

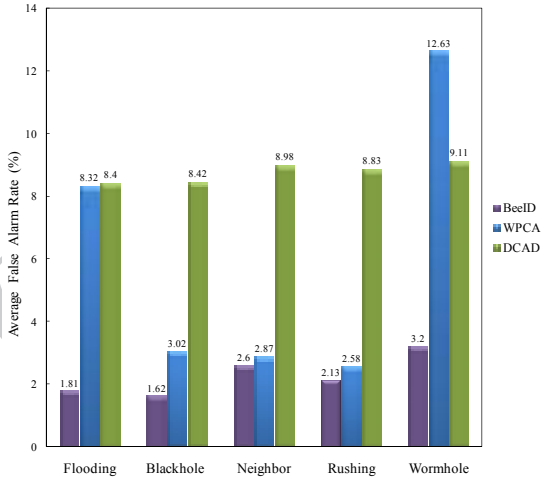
Table 6 compares the average running time of the detection and updating phases in each time-window for BeeID, DCAD, and WPCA. As shown in the table, the average running time of BeeID is 397.96(ms), which is slightly more than that of the two other approaches. Notice that the length of each time-window was set to $\Delta T = 100$(s) or 100000(ms). Hence, the average running time of BeeID is negligible with respect to ΔT, and it is acceptable in MANETs.

7 Conclusion and Discussion

A traditional approach to detect malicious network activities is to build a profile of the normal network traffic and then identify an activity as suspicious if it deviates from this profile. As the network topology in MANETs constantly changes over time due to the node mobility, the simple use of a static profile is not efficient. In this paper, we presented a dynamic hybrid approach based on the ABC and NS algorithms, called BeeID, for intrusion detection in AODV-based MANETs, which is able to adapt itself to the rapid changes of the network topology. The approach consists of three phases: training, detection, and updating. In the training phase, every node collects a set of feature vectors of its own normal network traffic and then applies the NicheNABC algorithm on them to generate a set of mature negative detectors to cover the



(a) Average detection rate



(b) Average false-alarm rate

Figure 11. Comparison of the performance of BeeID with that of DCAD [5] and WPCA [24]

nonself space. Each negative detector is a hypersphere with a variable radius. The Monte Carlo integration is used to estimate the coverage of the nonself space. In the detection phase, the node uses mature negative detectors to detect malicious network activities. In the updating phase, it updates mature negative detectors by one of two methods of partial updating or total updating. The Monte Carlo integration is used to determine when the partial updating or total updating should be done.

We conducted MANET simulations using the NS2 simulator and considered scenarios for detection of Flooding, Blackhole, Neighbor, Rushing, and Wormhole attacks. The performance of BeeID was measured using detection rate (DR) and false-alarm rate (FAR). We compared the performance of BeeID with that of DCAD [5] and WPCA [24] for detection of the routing attacks. The experimental results demonstrated that BeeID is able to make a better balance between

the detection rate and false-alarm rate, so that it increases the average detection rate by more than 0.85% and 8.71% and decreases the average false-alarm rate by more than 6.48% and 3.61% of the corresponding values of DCAD and WPCA, respectively. However, WPCA has a low detection rate for some routing attacks, such as Wormhole attack.

We discussed the effect of different values of control parameters, such as the radius of positive antigens, the length of a time-window, and the period of the total updating, on the performance of BeeID. The optimal parameter values were then selected from parameter values that made a better balance between the detection rate and false-alarm rate. In order to evaluate the effect of the partial updating on the amount of overlap between mature negative detectors and positive antigens, we calculated the average number of collisions between them before and after performing this process. The experimental results demonstrated that the partial updating reduces the average number of collisions between mature negative detectors and positive antigens for the routing attacks. On average, there was a 69%, 35%, 72%, 32%, and 47% reduction in the number of collisions for Flooding, Blackhole, Neighbor, Rushing, and Wormhole attacks, respectively.

The removal of old normal feature vectors and the lack of generating new mature negative detectors lead to a gradual decrease in the coverage of the nonself space by existing mature negative detectors. Hence, in some time-windows, we need to perform the total updating to generate new negative detectors and replace those with old negative detectors. The Monte Carlo integration was used to estimate the coverage of the nonself space and to determine when the total updating should be done.

Acknowledgment

This work was supported in part by the Iran Telecommunication Research Center (ITRC) under contract 90-01-04.

References

[1] P. M. Jawandhiya, "A Survey of Mobile Ad Hoc Network Attacks", *International Journal of Engineering Science and Technology*, 2(9):4063–4071, 2010.

[2] C. E. Perkins, E. M. B. Royer, and S. R. Das, "Ad Hoc On-demand Distance Vector (AODV) routing", in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, USA, 1999.

[3] I. Chlamtac, M. Conti, and J. N. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges", *Ad Hoc Networks*, 1(1):13–64, 2003.

[4] B. Wu, J. Chen, J. Wu, and M. Cardei, "A Survey of Attacks and Countermeasures in Mobile Ad Hoc Networks", in Y. Xiao, X. Shen, and D.-Z. Du (Eds.): *Wireless/Mobile Network Security*, Springer, pp. 103–135, 2006.

[5] M. Alikhany and M. Abadi, "A Dynamic Clustering-Based Approach for Anomaly Detection in AODV-Based MANETs", in *Proceedings of the 2011 International Symposium on Computer Networks and Distributed Systems (CNDS)*, Tehran, Iran, 2011.

[6] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-Nonself Discrimination in a Computer", in *Proceedings of the IEEE Computer Symposium on Research in Security and Privacy*, Los Alamitos, CA, USA, pp. 202–212, 1994.

[7] D. Dasgupta and L. F. Nino, *Immunological Computation: Theory and Applications*, Boca Raton: CRC Press, 2009.

[8] Z. Ji, "Negative Selection Algorithms: from the Thymus to V-detector", PhD Thesis, University of Memphis, 2006.

[9] B. Xu and Y. Zhuang, "Hybrid Detector Based Negative Selection Algorithm", in *Proceedings of the 2009 International Conference on Wireless Communications, Networking and Mobile Computing (WiCom)*, Beijing, China, 2009.

[10] D. Dasgupta, K. KrishnaKumar, D. Wong, and M. Berry, "Negative Selection Algorithm for Aircraft Fault Detection", in *Proceedings of the International Conference on Artificial Immune Systems (ICARIS)*, Catania, Italy, pp. 1–14, 2004.

[11] Z. Ji and D. Dasgupta, "Real-Valued Negative Selection Algorithm with Variable-Sized Detectors", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Seattle, WA, USA, 2004.

[12] D. Dasgupta and F. Gonzalez, "An Immunity-Based Technique to Characterize Intrusions in Computer Networks", *IEEE Transactions on Evolutionary Computation*, 6(3):281–291, 2002.

[13] D. Wang, F. Zhang, and L. Xi, "Evolving Boundary Detector for Anomaly Detection", *Expert Systems with Applications*, 38(3):2412–2420, 2011.

[14] D. Karaboga and B. Basturk, "On The Performance of Artificial Bee Colony (ABC) Algorithm", *Applied Soft Computing*, 8(1):687–697, 2008.

[15] P.-W. TSai, J.-S. Pan, B.-Y. Liao, and S.-C. Chu, "Enhanced Artificial Bee Colony Optimization", *International Journal of Innovative Comput-*

ing, Information and Control, 5(12):5081–5092, 2009.

[16] D. Karaboga and C. Ozturk, "Fuzzy Clustering with Artificial Bee Colony Algorithm", *Scientific Research and Essays*, 5(14):1899–1902, 2010.

[17] F. Gonzalez, D. Dasgupta, and L. F. Nino, "A Randomized Real-Valued Negative Selection Algorithm", in *Proceedings of the 2nd International Conference on Artificial Immune Systems*, Edinburgh, UK, pp. 261–272, 2003.

[18] P. Papadimitratos and Z. J. Haas, "Secure Routing for Mobile Ad Hoc Networks", in *Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS)*, San Antonio, TX, USA, 2002.

[19] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields., and E. M. Belding-Royer, "A Secure Routing Protocol for Ad Hoc Networks", in *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP)*, Paris, France, 2002.

[20] M. Ostaszewski, F. Seredynski, and P. Bouvry, "Immune Anomaly Detection Enhanced with Evolutionary Paradigms", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Seattle, WA, USA, 2006.

[21] S. Balachandran, D. Dasgupta, and L. Wang, "A Hybrid Approach for Misbehavior Detection in Wireless Ad Hoc Networks", in *Proceedings of the 1st Annual Symposium on Information Assurance*, New York, USA, 2006.

[22] S. Sarafijanovic, J. Boudec, "Secondary Response for Misbehavior Detection in Mobile Ad Hoc Networks", *IEEE Transaction on Neural Networks*, 16(5):1076–1087, 2005.

[23] X. Hang and H. Dai, "Applying both Positive and Negative Selection to Supervised Learning for Anomaly Detection", in *Proceedings of Genetic and Evolutionary Computation (GECCO)*, Washington DC, USA, 2005.

[24] H. Nakayama, S. Kurosawa, A. Jamalipour, Y. Nemoto, and N. Kato, "A Dynamic Anomaly Detection Scheme for AODV-Based Mobile Ad Hoc Networks", *IEEE Transactions on Vehicular Technology*, 58(5):2471–2481, 2009.

[25] D. Karaboga, An Idea Based On Honey Bee Swarm for Numerical Optimization, Technical Report, TR-06, Erciyes University, 2005.

[26] S. Taneja and A. Kush, "A Survey of Routing Protocols in Mobile Ad Hoc Networks", *International Journal of Innovation, Management and Technology (IJIMT)*, 1(3):279–285, 2010.

[27] D. B. Johnson, D. A. Maltz, and Y.-C. Hu, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)", Internet Draft, IETF MANET Working Group, 2003.

[28] V. Park and S. Corson, "Temporally-Ordered Routing Algorithm (TORA)", Functional Specification, Internet Draft, IETF MANET Working Group, 2001.

[29] X. Wang and J. Wong, "An End-to-end Detection of Wormhole Attack in Wireless Ad Hoc Networks", in *Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC)*, 1(1):39–48, 2007.

[30] G. S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, New York: Springer-Verlag, 1995.

[31] T. Stibor, J. Timmis, and C. Eckert, "On the Use of Hyperspheres in Artificial Immune Systems as Antibody Recognition Regions", in *Proceedings of 5th International Conference on Artificial Immune Systems (CARIS)*, Oeiras, Portugal, Springer-Verlag, vol. 4163, pp. 215–228, 2006.

[32] NS2—The Network Simulator, http://www.isi.edu/nsnam/ns/.

Fatemeh Barani received the B.Sc. degree in computer engineering from Ferdowsi University of Mashhad in 2008 and is currently a M.Sc. student of computer engineering at Tarbiat Modares University. Her main research interests are network security, intrusion detection, evolutionary computation, and swarm intelligence. Currently, she works on her thesis on anomaly detection in mobile ad hoc networks using negative selection and artificial bee colony algorithms.

Mahdi Abadi received the B.Sc. and M.Sc. degrees in computer engineering from Ferdowsi University of Mashhad in 1998 and Tarbiat Modares University in 2001, respectively. He also received the Ph.D. degree from Tarbiat Modares University in 2008, where he worked on the network vulnerability analysis. Since 2009, he has been an assistant professor in the Faculty of Electrical and Computer Engineering at Tarbiat Modares University. His main research interests are network security, intrusion detection and prevention, malware detection and analysis, evolutionary algorithms, and data mining.