A Hybrid Artificial Immune System (AIS) Model for Power Aware Secure Mobile Ad Hoc Networks (MANETs) Routing Protocols

N. Mazhar*,[a], M. Farooq[b]

[a]National University of Sciences and Technology (NUST), Sector H-12, Islamabad, Pakistan
[b]Next Generation Intelligent Networks Research Center (nexGIN RC), National University of Computer and Emerging Sciences (FAST-NUCES), Islamabad 44000, Pakistan

Abstract

Securing ad hoc routing protocols for MANETs is a significant challenge due to number of reasons: (1) mobility results in continuously changing network topology – the premise of stable *self* or *non-self* is void, (2) the proposed security solution must be lightweight so that it can be deployed on resource constrained mobile nodes, and (3) the solution should provide high detection accuracy and low false positive rate. The major contribution of this paper is a hybrid AIS model – combining the relevant features of classical *self/non-self* paradigm with the emerging *danger theory* paradigm – that has the capability to meet the above-mentioned challenges of the MANET environment. As a case study, we use our hybrid model to develop a power aware security framework for *BeeAdHoc* – a well-known bio-inspired routing protocol. We have realized our framework in ns-2 simulator. We have also developed an attacker framework in ns-2 that has the capability to launch a number of Byzantine attacks on *BeeAdHoc*. The results of our experiments show that our proposed framework meets all its requirements: (1) the adaptive learning because of changing *self/non-self*, (2) high detection accuracy and low false positive rate, (3) lightweight in terms of processing and communication overheads, and (4) better or comparable performance compared with non-secure versions of existing state-of-the-art MANET routing protocols – *DSR* and *AODV*. We have also compared our hybrid AIS model with *self/non-self*, *danger theory* and a conventional anomaly detection system to show its merits over these schemes. Finally, we propose an extension of the framework for securing *DSR*.

Key words: Artificial Immune Systems, dendritic cells, misbehavior detection, mobile ad hoc networks

1. Introduction

Researchers and industrial experts are now devoting significant amount of their efforts to realize *Ubiquitous Computing* [1][2] and *Pervasive Computing* [3][4] paradigms – though the concepts emerged approximately two decades ago. An important enabler for these paradigms is Mobile Ad Hoc Networks (MANETs), which are becoming an active area of research [5]. A MANET simply consists of a set of wireless mobile nodes that can quickly organize into a network – without the need for an infrastructure – and start communicating using their wireless links. The nodes randomly move in a given theater; as a result, the network topology keeps on changing. These ad hoc networks have found numerous applications in civil and military domains[6]. Since these MANETs are vulnerable to a number of security threats, hence it becomes pertinent to provide a secure computing environment for MANETs [7].

Most recently proposed Intrusion Detection Systems (IDS) [8] have the ability to detect zero-day previously unseen attacks. The IDS learns the benign *self* behavior of the system during an initial *learning phase*. Later during the protection phase, any deviation from the *self* is classified as *non-self* – a sign of anomaly/intrusion. This paradigm, however, works on the premise of a stable *self*. Artificial Immune Systems (AISs) have served as a natural source of inspiration for designing this type of network anomaly detection systems [9] [10][11][12][13]. An interested reader is referred to [14][15] for a comprehensive review. Most of these systems are based on *self/non-self* model of Biological Immune System (BIS) in which the adaptive immune response is elicited through recognition of foreign entities (pathogens) in the body. The *self* here is the set of antigens presented to the immune system early in life; as a result, the model assumes that organisms learn and live with a *"static self"* throughout their life.

Note that in MANET routing, however, the routes frequently change because of mobility of nodes; as a result, the existing valid routes might become unavailable and the new legitimate routes might become available. In such an environment, it becomes a challenging task to differentiate between new legitimate routes, and new illegitimate routes – forged through tampering or fabrication attacks – by the malicious nodes. Consequently, the above-mentioned premise of a stable *self* does not hold in MANETs; as a result, we need to revisit the above-mentioned AIS paradigm – learning a *static self* and raising an alarm in case of any deviation from it – in the scenario of pro-

*Corresponding author
Email addresses: naumaz@yahoo.com (N. Mazhar), muddassar.farooq@nu.edu.pk (M. Farooq)

viding security in MANETs. The following scenarios are often experienced in MANETs that directly effect the design of the AIS based power aware security framework:

1. **New benign behavior.** Due to mobility, new legitimate routes *(self)* become available which are never observed during the learning phase.
2. **Benign behavior turns malicious.** The malicious nodes start advertising the legitimate routes – learnt during the learning phase – which are currently not available.
3. **Malicious behavior turns benign.** The routes that are considered part of *non-self* happen to become available as legitimate routes.
4. **New malicious behavior.** The routes that are never observed in the learning phase become part of *non-self*.

We can conclude from the above four scenarios that *self/non-self* paradigm, utilizing the notion of learning a *"static self"*, will work only for the last scenario. In two cases – "new benign behavior" and "malicious behavior turns benign" – the static paradigm will result in false positives; while in case of "benign behavior turns malicious " it will result in false negatives.

Matzinger has shown that changes do happen in a human body – tumors, transplants etc – and the *self/non-self* model does not adequately explain these changes [16][17]. The danger theory [18] provides an alternative paradigm for developing network Anomaly Detection Systems (ADS) by taking inspiration from the working of the innate immune system. The proponents of this theory believe that the activation of adaptive immune response requires the presence of "danger" in the tissues in addition to pathogen recognition. The tissue context is "dangerous" only when the body cells are damaged due to a pathogenic infection. The damage is determined by the Dendritic Cells (DCs) of the innate immune system. The danger theory paradigm, therefore, views the innate immune system as an adaptive controller of the adaptive immune response [19][20]. A recent example of this paradigm is Dendritic Cell Algorithm (DCA) [21][22] that is shown to perform anomaly detection [23]. But the algorithm detects "danger" instead of the *non-self*. Such a system, in which we dynamically determine the context of a cell, is a better candidate to cater for changing *self/non-self*.

In addition to the above-mentioned challenges, our proposed solution must be able to meet following performance requirements, if we want to deploy it on real-world resource constrained MANET nodes:

- ***High detection accuracy*** must be achieved by reducing the high false positive rate due to changes in the network topology;
- ***Small detection delay*** is critical in MANET environment;
- ***Minimum control overhead***[1] is a must to detect anomalies in an energy efficient manner (transmission/reception of packets from/to a wireless interface is the most dominating source of energy depletion);
- ***Minimum processing overhead*** is also a desirable feature for energy efficient anomaly detection;
- ***High network performance*** of the non-secure version of the protocol must be guaranteed by the secure version.

The other option for providing security in MANETs is to utilize the classical cryptographic systems that are proven to have high control and processing overheads [24][25]. The well-known cryptography based solutions – *ARIADNE* [26], which uses symmetric cryptography to secure *DSR* protocol [27], and Secure Ad-Hoc On-demand Distance Vector *(SAODV)* [28][29], which uses asymmetric cryptography for security of *AODV* [30] – demand secure key distribution mechanisms and have high processing overheads, especially in case of asymmetric systems. Moreover, these approaches require computing digital signatures or message hashes – a step that has significant processing overheads [24] – which must be transmitted within the control packets; as a result, the control overhead also significantly increases. As a consequence, the effective throughput of the protocol [25] [24] is significantly reduced.

To conclude, we cannot utilize heavyweight conventional asymmetric digital signature based cryptographic solutions to detect misbehaving/malicious nodes in a MANET environment. Similarly, using stand alone *self/non-self* paradigm is not suitable for MANETs. Therefore, in this paper, we propose our integrated model for AIS – *iAIS* – which links the innate immune system with the adaptive immune system through T-helper cells. To the best of our knowledge, this is the first effort in which relevant features of both models are combined to cater for challenging MANETs environments. The relevant features of our proposed *iAIS* model are:

1. **Adaptive detector database.** Unlike the fixed initialization and learning phases in the *self/non-self* AIS, the detector database in *iAIS* is not static; rather, it temporally evolves with experience. This continuous feedback is important for learning because the learnt definition of *self* may be volatile – *self* may change over time. Therefore, by mapping the role of DCs and T-helper cells in the co-stimulation of B cells, we devise an evolving and adaptive detector population.
2. **Combined innate and adaptive immune response.** The response produced by *iAIS* is a combined response from danger theory based AIS (innate response) and negative selection based AIS (adaptive response). The output response from DCA is translated into the negative selection based AIS with the help of T-helper cells. The danger theory concepts are used to determine the context of an antigen and the adaptive immune system then performs pattern matching through B-cells. T-helper cells are activated by DCs to provide co-stimulation of B-cells for detection of *non-self* antigens. DCs have a predefined individual threshold for every input signal in DCA. The values of these thresholds are set optimistically for approximately 100% true positive rate. This is important to avoid the noise in learning phase of negative selection based AIS.

[1]The portion of the bandwidth occupied by the control packets

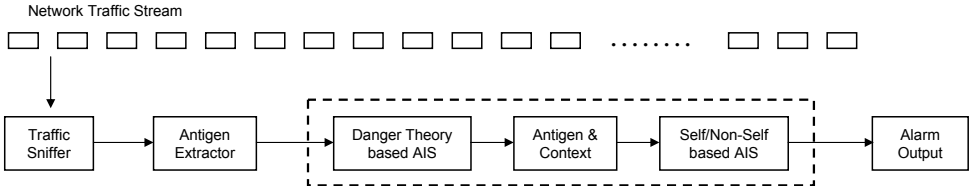


Figure 1: Application model of an ADS using *iAIS*

We further simplify our *iAIS* model to make it energy efficient – reducing processing overheads – and suitable for deployment on battery constrained mobile nodes. The new model – *iLite* – removes the need of T-helper cells and instead allows DCs to directly co-stimulate B-cells for detecting *non-self* antigens. We use *iLite* to develop *iBeeAIS* – a security framework for a bio-inspired MANET routing protocol, *BeeAdHoc* [31]. We have implemented *iBeeAIS* in ns-2 and the results of our experiments show that *iBeeAIS* not only provides the required security against routing attacks by malicious nodes but also achieves the desired network performance. Remember that the security framework of a routing protocol can increase its control overhead, reduce throughput, and increase the node's energy consumption. Therefore, it is important to study - under no threats or attacks - the deterioration in the protocol's network performance. The network performance of *iBeeAIS* is found to be comparable to *BeeAdHoc* protocol and significantly better when compared with non-secure classical MANET routing algorithms: *DSR* and *AODV*. This validates our thesis that combining innate and adaptive immunity results in a robust security framework for time-varying adaptive environments.

A high level application model of an ADS using our proposed *iAIS* is shown in Fig. 1. The ADS is modeled as a traffic filter, which may be customized and scaled to appropriate network point such as a border router. The traffic sniffer sniffs packets from the network stream from which antigens are extracted. The danger theory based AIS computes traffic/packet features to be used as signals to determine the biological context of "dangerous" or "safe". The antigen along with its context is then presented to the *self/non-self* based AIS, which distinguishes the *non-self* from *self* to generate an ADS alarm.

The rest of the paper is organized as follows. In Section 2, we provide a brief introduction to the *BeeAdHoc* protocol. We then introduce the two basic AIS models for B-cells and DCs in Section 3. In Section 4, we discuss in detail the design of our integrated AIS model – *iAIS* – which combines the *self/non-self* discrimination with the context of an antigen determined by the danger theory. The *iAIS* can be simplified to lower its computational complexity for MANET environment. So in Section 5, we introduce *iLite* that uses pre-activated B-cells. Section 6 describes in detail the design and implementation of our proposed security framework, *iBeeAIS*, which is an implementation of the *iLite* model in ns-2 to secure the *BeeAdHoc* routing protocol. Section 7 describes our attack simulations to test the ability of *iBeeAIS* to counter the routing attacks and tabulate results for the performance of *iBeeAIS* in terms of detection rate and detection delay. We then compare *iBeeAIS* with *self/non-self* and danger theory based systems in Section 8; the aim is to show the benefit of hybrid AIS over other approaches. In Section 9, we provide results of our extensive network simulations, comparing *iBeeAIS* with other MANET protocols. Section 10 discusses related research in the area and in Section 11 we present an adaptation of *iAIS* to secure the *DSR* protocol. Finally, we conclude the paper with an outlook to our future research.

2. BeeAdHoc Protocol and its Security Analysis

BeeAdHoc [31] is a Bio/Nature inspired, reactive, source routing protocol for MANETs. It has a *Bee Agent Model* inspired from the foraging principles of honey bees [32][33][34]. The agent model mainly utilizes *Scouts* to discover new routes and *Foragers* to carry data from source to destination. When route to a destination is needed, the source node broadcasts a *forward scout* in the network. The intermediate nodes receiving the scout, append their addresses to the source route and rebroadcast the scout until it reaches at the destination. The destination node then reverses the source route, creating a *backward scout*, and sends it back to the source. At the source node, the route is advertised to other foragers. Consequently, the foragers select one of the routes and use it to transport data to the destination. On their way, foragers also collect the routing information that is used to calculate the forager dance number, which represents the quality of the path traversed. A route's quality influences the future selection of the path by foragers; path with a better quality metric has higher probability of being selected. This section contains only a brief description of the *BeeAdHoc* protocol; interested readers can find details in [31][35][36].

2.1. BeeAdHoc Vulnerabilities

The security threat analysis of *BeeAdHoc* in [25] has shown vulnerabilities in the protocol that could be exploited by a malicious node to launch a number of *Byzantine* [37] attacks – disrupting the normal routing behavior of *BeeAdHoc*. We outline few of the attacks here.

Scout related attacks. A malicious node can modify the source route in scouts or it can forge a scout by spoofing its source address or inserting fake scout ID, or both.

Forager route related attacks. Foragers carry data packets in their payload and are transmitted as unicast packets. A malicious node can tamper with the source route of a forager or launch a forged forager with a spoofed source address and source route towards the destination.

Forager route info related attacks. A malicious node can modify the routing parameters, carried by foragers, to artificially enhance the quality of a path; as a result, the malicious node can divert data traffic on a low quality route, which ultimately reduces the overall network throughput.

The tampered or forged bee agents result in establishing fake routes, which could severely degrade the performance of the network. One fundamental assumption for successfully launching these attacks is that the malicious node has sniffed valid routes from a source to a destination. If a hop-by-hop connectivity is not guaranteed in the fake source route, the forged scout or forager will subsequently be dropped. In a worst case scenario this could degenerate into a complete DoS attack because no foragers or scouts will ever return at the source node.

3. Network Anomaly Detection: The Basic AIS Models

In this section we describe the two AIS models that have been utilized earlier in the domain of network anomaly detection. We briefly explain the working of these two models to help the reader in better understanding of our security framework.

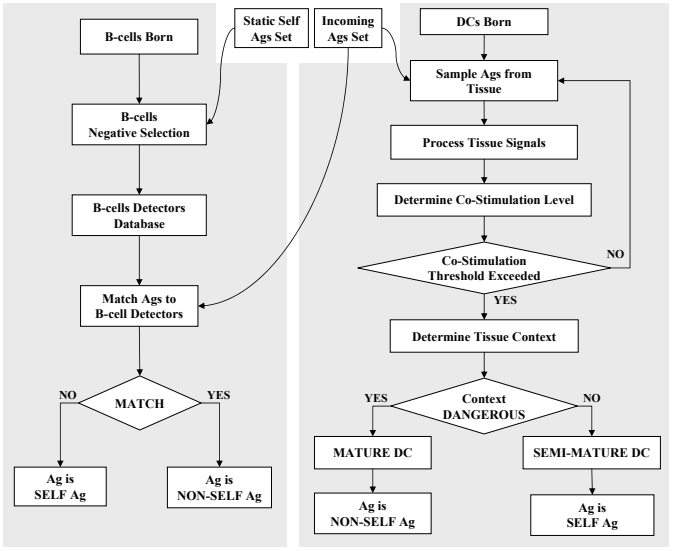
3.1. Basic Model for B-cells

This model (Fig. 2(a)) presents B-cells in the classical context of *self non-self* discrimination paradigm. The model works in two phases; (1) learning and (2) operation. In *learning phase*, the model learns the benign behavior of a given system. In the *operational phase*, it receives antigens (Ags) and classifies them as *self* or *non-self*. The model utilizes two sets of Ags:

- **Static self Ags set.** The set of Ags presumed to be *self* are used to train the model during the learning phase. We call this set *"static"* because it represents the system state only for a short span of time at system start-up and not the complete system dynamics in the form of behavioral changes that may occur in the system later in its operational life. The set is used for *negative selection* of B-cells.

- **Incoming Ags set.** The system has to classify incoming Ags seen by the model during its operational phase.

B-cells Detectors Database. The basic B-cells model generates a set of B-cells detectors that have the ability to detect *non-self* Ags. This detector set, however, provides limited coverage of *non-self* space because of holes; as a result, the system has high false negatives. Secondly, the B-cells detectors generated by this model can only be used to detect anomalies for a system with *"static self"* – a *"self set"* that does not change the system's behavior during its lifespan. This is due to the fact that the system learns its normal behavior only once during the initial learning phase. It lacks the ability to adapt to a changing *self* by modifying its detector set based on the feedback from the system.



(a) Basic model of B-cells (b) Basic model of the Dendritic cells

Figure 2: Modeling the B-cells and the Dendritic cells

3.2. Basic Model for Dendritic Cells

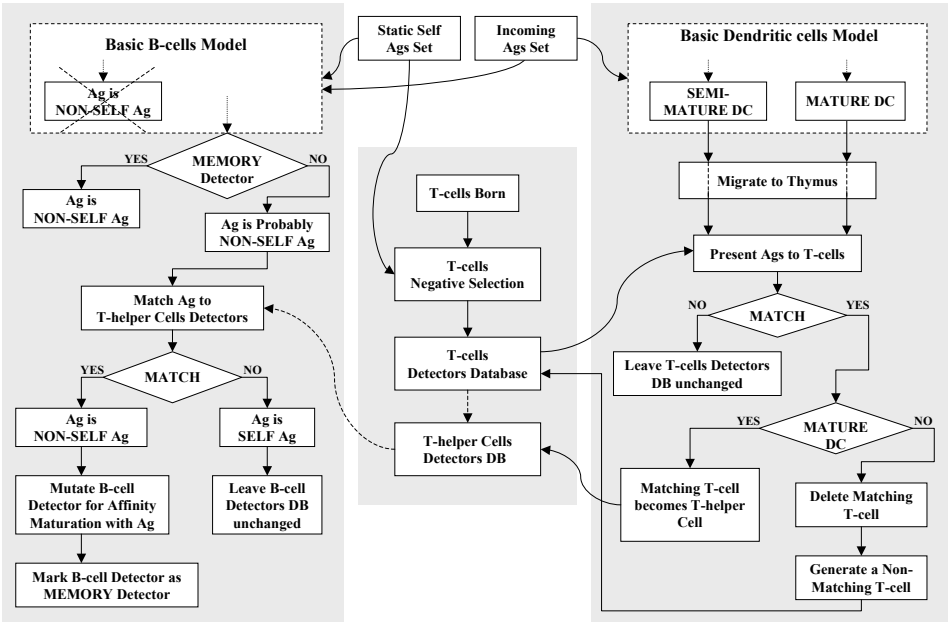
The basic model for DCs depicts the functional behavior of DCs starting from Ag sampling in tissues till the determination of tissue context as "safe" or "dangerous". The DCs process signals present in the tissue at the time of sampling, (Fig. 2(b)), and determine the *co-stimulation* level. If the *co-stimulation* threshold has exceeded, the DCs determine the tissue's context and make a transition either to a mature state if the context is "dangerous" or to semi-mature state if the context is "safe". This working principle, in essence, is similar to the DC behavior modeled in the DCA [23]. The difference, however, is that our system does signal processing at a relatively higher level. The Ag is declared as *self* if the DC differentiates to a semi-mature state, and as *non-self* if it enters the mature state. It is to be noted that DCA does not present the sampled Ags to T-cells in *Thymus* to activate/de-activate the T-cells. This function is, however, included in our model that we present in the next section.

4. *iAIS*: An Integrated AIS Model

We have already discussed that neither *self/non-self* nor danger theory paradigm can successfully address requirements of secure routing in MANETs. Therefore, we prepare to join relevant concepts of both paradigms to develop an anomaly detection system for MANETs. Fig. 3 shows the design of our new *iAIS* model. To the best of our knowledge this is the first attempt in AIS community to do anomaly detection by combining concepts of B-cells, T-cells and DCs in a unified system.

4.1. Extending the Basic DC Model and Linking with T-cells

The basic dendritic cell model from Section 3.2 can be extended to include: (1) the presentation of sampled Ags by DCs in *thymus* and (2) the maturation/activation of T-cells. The motivation of these enhancements is to:



(a) Extended B-cells model (b) T-cells model (c) Extended Dendritic cells model

Figure 3: *iAIS*: An Integrated AIS Model for Activation of B-Cells through Dendritic Cells and T-helper Cells.

- Dynamically learn the changes in the system *self* based on the feedback from DCs.

- Generate a database of T-helper cells detectors that can provide co-stimulation to B-cells.

The model of our proposed system is shown in Fig. 3(c). Once the DCs have selected one of the two differentiation pathways – mature or semi-mature – they migrate to *thymus* and present their sampled Ags to T-cells. T-cells are generated using negative selection algorithm (Fig. 3(b)). The negative selection of T-cells employs a set of *"static self"* Ags, representing the system state during learning phase. In *thymus*, if a T-cell matches an Ag presented by a semi-mature DC, the T-cell dies. This helps in incorporating the changing *self* information in the system. By deleting T-cells we provide the system tolerance to the new *self*. On the other hand, in case of interaction with a mature DC, the matching T-cell gets activated to become a T-helper cell. In this way, the DCs generate a set of T-helper cells detectors that are capable of co-stimulating the B-cells.

T-helper Cells Detectors Database. During sampling, the DCs can pick up any Ags – *self* or *non-self* – from the tissue. Once they have processed the signals in the tissue, a DC presents all sampled Ags in the same context i.e. either "safe" or "dangerous". Consequently, it is possible for a DC to present some of the *self* Ags in the "dangerous" context. To eliminate these *self* Ags in a wrong context from mature DCs, the DC Ags are also matched with the negatively selected T-cells. Since negative selection ensures that *self* reactive T-cells are deleted, any matching DC Ags are, expectedly, truly *non-self*. The resultant activated T-helper cells can then be used effectively to co-stimulate B-cells and to trigger an adaptive immune response.

4.2. Extending the Basic B-cells Model, with Linkage to Dendritic Cells and T-cells

Our proposed model also extends the basic B-cells model from Section 3.1 and links its functionality to the extended dendritic cells and the T-cells model described in Section 4.1. The basic B-cells model classifies the incoming Ags as *self* or *non-self* after matching them with the negatively selected B-cells detectors. We delay the decision to classify an Ag as *non-self*, (see Fig. 3(a)). Initially, our model classifies the matching Ags as "*potentially non-self*", requiring further co-stimulation from the T-helper cells. And this can happen only if the B-cell detector is not a memory detector because a matching Ag for memory detectors is immediately declared as *non-self* to improve the secondary response time of the system.

The Ag declared as "*potentially non-self*" is matched with the detectors in the T-helper cells detector database. If there is a match, the respective T-helper cell provides co-stimulation signal to B-cells to initiate the adaptive immune response. This provides close emulation of the BIS, which requires co-stimulation to reduce the chances of accidentally reacting to *self* because of two reasons: (1) the *self* changes over time, and (2) new and potentially *self* reactive lymphocytes are produced when B-cells undergo hypermutation during clonal selection.

The basic B-cells model learns only an initial *"static self"*, which is likely to change with time in a MANET scenario. In our model, we use co-stimulation for making the final classification decision that prevents the changed *self* from being misinterpreted as *non-self*; as a result, false positives are reduced. We incorporate a limited version of clonal selection for the matching B-cell detectors. The B-cell detectors undergo mutation to achieve affinity maturation with the *non-self* Ags. These detectors, therefore, develop into memory detectors having high affinity with the matching *non-self* Ag.

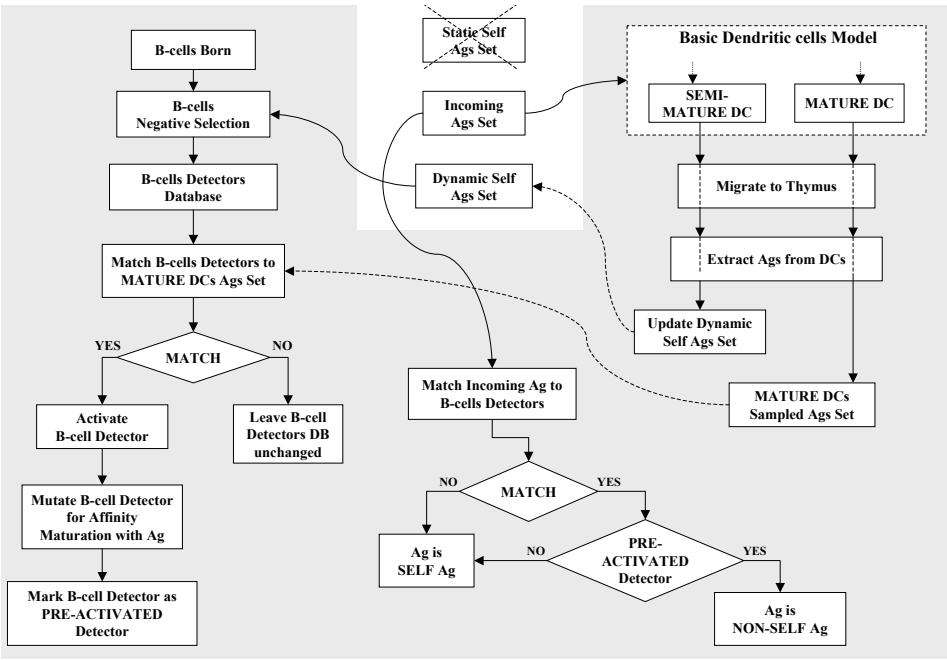


Figure 4: *iLite*: A Lightweight AIS Model using Pre-Activated B-cells

5. *iLite*: Lightweight AIS Model with Pre-Activated B-cells

The *iAIS* model of Fig. 3 follows very closely the contours of the BIS. Our investigation reveals that we can develop a simple and efficient ADS for MANETs if we do not do a 1–1 mapping of the biological concepts to ADS. In this context, we propose a lightweight ADS framework – *iLite* – in Fig. 4. The system utilizes the slightly modified versions of the extended B-cells and the dendritic cells models introduced in Section 4. The notable feature of this model is that the B-cells are directly linked with the DCs. We do not use the T-helper cells to stimulate the B-cells; furthermore, we use the two differentiation states of the DCs to perform the following two functions in *thymus*:

- The Ags presented by Semi-mature DCs are used to learn a *dynamic self Ags* set, i.e, new Ags, which we can safely declare as *self* based on the feedback from the DCs.
- The Ags presented by Mature DCs are used to pre-activate the B-cells directly without the need of T-helper cells.

Dynamic Self Ags Set. In systems with a changing *self*, a set of static *self* Ags does not provide significant help. Therefore, we need to learn the new *self* as state of the system changes. The dynamic *self* Ags set of this model represents the most recent system state and is generated using the feedback from the DCs. When DCs determine that the tissue's context is "safe", they present the sampled Ags as *self* Ags in *thymus*. Our model uses this *self* for negative selection of B-cells.

Pre-Activated B-cells Detectors. When DCs present Ags in a mature context, they are sampled with a high probability as *non-self* Ags in the presence of "danger". To segregate these *non-self* Ags from the *self* ones, we match all sampled Ags with the negatively selected set of B-cell detectors and prime only those B-cells that show high affinity. This allows pre-activation of B-cells before they are exposed to the incoming Ags set. We also mutate the primed B-cells detectors to further enhance affinity with the *non-self* Ags presented by mature DCs. The resulting detectors, therefore, should have a high potential to detect the *non-self* Ags.

We now demonstrate that our proposed model can be used to develop a simple, efficient and lightweight security framework for the *BeeAdHoc* MANET routing protocol.

6. *iBeeAIS*: An AIS Security Framework for *BeeAdHoc* Protocol Using the *iLite* Model

In this section, we present the design and implementation of a integrated AIS security framework, *iBeeAIS*, for misbehavior detection in MANET routing protocol, *BeeAdHoc*. *iBeeAIS* is based on the *iLite* model of Section 5. The new framework has been temporally evolved after proposing the *BeeSec* [25], *BeeAIS* [38] and *BeeAIS-DC* [40] security frameworks.

In this framework, the Ags in a tissue are sampled by DCs and then the tissue context is determined as "safe" or "dangerous". This feature enables dynamic learning of the system *self* and *non-self*. It then links the DCs directly to B-cells to create a set of pre-activated B-cells detectors that can mutate and achieve high affinity with the *non-self* Ag. Therefore, using our proposed *iLite* model, the *iBeeAIS* is able to achieve good detection accuracy with low false alarm rates for the forager related attacks, which is not possible with *BeeAIS-DC* [40].

iBeeAIS also employs the concept of tissue "inflammation". The presence of inflammation accelerates the immune response for DC migration without the need for co-stimulation. In BIS, B-cells remain active only for a short duration and then go back

Table 1: List of symbols used in description of *iBeeAIS* algorithm

Symbol	Description
IP_{src}, IP_{dst}	source & destination Internet Protocol (IP) addresses
FS_{sd}, BS_{ds}	forward & backward scout between nodes s & d
Ag, Ag_{ns}	antigen, & *non-self* antigen
DC_{sct}, DC_{fgr}	scout & forager dendritic cells
BC_{sct}, BC_{fgr}	scout & forager B-cells
$LifeAct_{bc}$	active life assigned to B-cell when it matches a *non-self* Ag presented by mature DC
$PathUseEff_s$, $PathUseEff_l$	path use efficiency of the shortest or the longest route followed by foragers during the last UDINT period
$FlagInfl_{sct}$, $FlagInfl_{path}$, $FlagInfl_{rtInfo}$	inflammation flags for scout & forager DCs that are raised when a DC crosses the co-stimulation level, indicating damage to tissues
$RtLen$, $RtInfo$	length of source route & route information carried in forager packet header to determine route quality
S_{sct}, D_{sct}	source & destination of a scout
T_{curr}, D_{ham}	current time during simulation & hamming distance
$Count_{FS}$, $Count_{BS}$	number of forward & backward scouts received
$CountFgr_{dc}$	number of foragers received
$SumRtInfo_{dc}$	sum of route info values of foragers following a specific path in one UDINT period of time
$AvgRtInfo_{dc}$, $AvgRtInfo_{bc}$	average route info value of foragers following a specific path computed after each UDINT period for a DC_{fgr} and stored in a BC_{fgr}
$FlagAct_{sct}$, $FlagAct_{path}$, $FlagAct_{rtInfo}$	activation flags for scout & forager BCs that are raised when a B-cell matches a *non-self* Ag presented by mature DC

to their de-activated state; likewise, *iBeeAIS* assigns an activation life to B-cells. It is only when B-cells are activated that they can match and detect *non-self* Ags. The complete functioning of *iBeeAIS* system is now being described.

6.1. Dendritic Cells

A DC is born when a node sees a scout or a forager for the first time. *iBeeAIS* uses two types of dendritic cells, the *scout dendritic cells* (DC_{sct}) and the *forager dendritic cells* (DC_{fgr}). The major attributes of these DCs include:

DC Ag. Ags are formed from the incoming network traffic (*forward* scout, *backward* scout or forager) to represent sampling of Ags by DCs. The Ag format selected for *iBeeAIS* is shown in Fig. 5, which is basically similar to Ag format used in *BeeAIS* [38]. The minor differences are: (1) in case of scout Ags, the *"Scout ID"* gene is replaced with *"Scout Destination"*, and (2) in case of *foragers*, we have merged the two Ag types into a single one and the *danger signal* helps to differentiate source route modification from route information modification. *iBeeAIS* Ag has four genes with lengths of *16, 16, 4 and 16* bits,

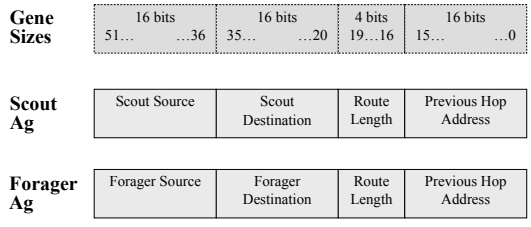


Figure 5: *iBeeAIS* Ag formats

concatenated to create the final 52 bits long bits string. Ags are represented in binary hamming shape space. *iBeeAIS* genes represent header field values extracted from the bee agents, as:

$$DC_{sct}\ Ag = \langle S_{sct},\ D_{sct},\ RtLen,\ node_{i-1} \rangle \quad (1)$$

$$DC_{fgr}\ Ag = \langle IP_{src},\ IP_{dst},\ RtLen,\ node_{i-1} \rangle \quad (2)$$

DC Life. DCs in our system always present the most recent system state. Therefore, at the time of birth, they are assigned a random short life. When a DC's life ends, it simply dies unless some relevant Ags arrive. The life of a new born scout dendritic cell is determined by Eq. 3. We can see that a DC_{sct} lives in the system for a maximum of two consecutive UDINT periods unless it gets a new life. In the case of DC_{fgr}, a new born is assigned life as in Eq. 4.

$$DC_{sct}\ life = T_{curr} + UDINT \quad (3)$$

$$DC_{fgr}\ life = (T_{curr} + rand()\ \%\ UDINT) + UDINT \quad (4)$$

Therefore, the life of a DC_{sct} is kept smaller compared to that of a DC_{fgr} for memory efficiency. The number of DC_{sct} in the system, especially during initial flooding phase, is significantly more compared with DC_{fgr} during normal system operation; as a result, they might take up a significant amount of memory.

DC State. The state of a DC may be *immature*, *semi-mature* or *mature*. At the time of birth, a DC is *immature*. When it samples the Ags and is exposed to *safe signals* it makes a transition to the *semi-mature* state. In comparison, if it is exposed to *danger signals* it differentiates to the *mature* state.

Tissue Context Data. DCs collect data that is required to determine tissue's context. The DC_{sct} keep a count for receiving the *forward* or *backward* scouts ($Count_{FS}$ or $Count_{BS}$) while DC_{fgr} is the count for foragers received on a particular path ($CountFgr_{dc}$). This information is used later by *iBeeAIS* to determine the occurrence of *danger signals*.

Co-stimulation. As in BIS, the DCs in our system need co-stimulation for migration. The DCs maintain and monitor the co-stimulation level and only declare the Ag context as "dangerous" when it exceeds the threshold value. This enables the DCs to experience a "danger" multiple times before declaring the tissue context as "dangerous".

Inflammation. Inflammation in BIS increases the speed of response to "dangerous" Ags. The DCs in our system, which cross the costimulation threshold, set their inflammation as "HIGH". These DCs no longer need further co-stimulation to determine their differentiation state as mature or semi-mature. But, this applies only to those DCs, which experience temporally close "dangerous" events within their lifetimes.

The forager DCs also collect additional information to determine the existence of *danger signals* pertaining to unauthorised modification of route information of a forager. This concept

has been borrowed from our earlier implementation of *BeeAIS* [38]. For this, a DC maintains the sum of the route information values obtained from all foragers following the same path in every update interval (UDINT) period of time. The value is computed as in Eq. 5, with new born DCs having initial value of $SumRtInfo_{dc}$ set to zero.

$$SumRtInfo_{dc} += T_{curr} - RtInfo_{fgr} \quad (5)$$

Table 2: List of *iBeeAIS* parameters

Parameter	Description
UDINT	fixed small interval of time defined for the system such that after each UDINT period the system processes the DCs to determine the tissue context, and processes the B-cells for *self* tolerance, activation and affinity maturation
THRESH_RCVD_FS THRESH_RCVD_BS	upper limit for average forward or backward scouts to be received by a node before context can be declared dangerous
FACTOR_PATH FACTOR_RTINFO	weighting factor between normal and anomalous path use efficiency and route information for foragers to declare the context as dangerous
CO_STIMUL_SCT CO_STIMUL_PATH CO_STIMUL_RTINFO	co-stimulatory threshold for transition of DC state from IMMATURE to MATURE or SEMIMATURE and presentation of sampled *non-self* Ag in thymus for B-cell activation
NUM_BC_SCT NUM_BC_FGR	number of detectors maintained by the system at any given time for matching the incoming scout or forager Ags
EPS_NS_BC_SCT EPS_NS_BC_FGR	cross reactivity threshold for negative selection of scout and forager B-cells
ACT_FCTR_SCT ACT_FCTR_FGR	distance of the scout or forager detector from the respective *non-self* Ag, for the detector to be selected for activation
SPEC_AB_SCT SPEC_AB_FGR	distance to be maintained between the scout or forager antibody and the respective *non-self* Ag for affinity maturation
DET_RANGE_SCT DET_RANGE_FGR	detection range of scout/forager B-cells antibodies that determines how far away from itself the detector can detect Ags

6.2. B-Cells (BCs)

In *iBeeAIS*, the B-cells provide a set of antibodies or detectors that can match and initiate the adaptive immune response against the perceived *non-self* Ags presented by mature DCs. B-cells have the ability to mutate their antibodies for an increased antigenic affinity and to generate a more specific response against a particular pathogen. *iBeeAIS* uses two types of B-cells; (1) the *scout B-cells* (BC_{sct}) and (2) the *forager B-cells* (BC_{fgr}). These B-cells have the following attributes:

BC Antibody. B-cells antibodies act as detectors and are used to match the incoming Ags. As in BIS, each BC in *iBeeAIS* has only one antibody and is therefore specific to only one type of Ag. Initially, the antibodies are generated randomly. Later during the system operation, the B-cells are periodically subjected to negative selection with respect to the *self* Ags presented by semi-mature DCs. The surviving antibodies can be activated for affinity maturation with the passage of time.

Activation Flag. The activation flags of B-cells indicate the "activated" state once they match with the *non-self* Ags presented by mature DCs. A BC_{sct} has only one flag to detect scout attacks, while a BC_{fgr} has two flags to detect the two forager related attacks as discussed in Section 6.3.

Active Life. B-cell is assigned an active life when it matches a potential *non-self* Ag presented by a mature DC. A B-cell can detect *non-self* Ags only during its active lifetime. Once the active life of B-cell is over, it is de-activated. During this state, it might again be activated if it survives the frequent negative selections. The active life of a B-cell is given as:

$$LifeAct_{bc} = T_{curr} + UDINT * (rand() \% 2 + 2) \quad (6)$$

6.3. Danger Signals for Scouts and Foragers

In *iBeeAIS*, the *danger signals* are computed at fixed periodic intervals of time with a (UDINT) period. Our analysis of *BeeAdHoc* protocol has revealed that we can encounter three types of *danger signals* related to scout and forager attacks.

6.3.1. Scout Related Danger Signal

While the system is in operation, at the end of every UDINT period, the DCs determine the average forward and backward scouts received on every path. Each time the computed averages exceed their respective thresholds (THRESH_RCVD_FS, THRESH_RCVD_BS), the co-stimulation level for that path is increased. Finally, when the co-stimulation level exceeds the co-stimulation threshold for scouts (CO_STIMUL_SCT), the context for this DC becomes "dangerous". At the same time, the inflammation flag – $FlagInfl_{sct}$ – for this DC is raised to indicate that the tissue is damaged. As long as the $FlagInfl_{sct}$ is "HIGH", the context for this DC within its remaining lifetime remains "dangerous". If the attack is not detected the co-stimulation level is gradually reduced to lower the $FlagInfl_{sct}$.

The Eq. 3 shows that a scout DC can survive a minimum of two consecutive UDINT time periods before its natural death. Within this period, however, if a similar scout arrives again, the DC life is increased for another UDINT period. This implies that the *danger signal* will be detected only if: (1) within the life span of a DC the *non-self* Ag keeps arriving, (2) Ag is detected as suspected, and (3) the number of UDINT periods within which the Ag is detected as suspected exceeds the threshold. This provides sufficient co-stimulation before raising the $FlagInfl_{sct}$ and helps to reduce the rate of false positives.

Once raised, the $FlagInfl_{sct}$ classifies the relevant *non-self* Ag as "dangerous" without any further need for co-stimulation. The $FlagInfl_{sct}$ – indicating the presence of *danger signal* – remains high for subsequent (CO_STIMUL_SCT + 1) number of UDINT periods, if the attack is no more detected. This helps in lowering the false negatives if the attack continues but skips detection in contiguous UDINT periods. A high value of CO_STIMUL_SCT would slow down the system's response and cause an increase in the false negative rate. The steps involved in computing the scout related *danger signal* are detailed in Algorithm-1.

6.3.2. Forager Path Related Danger Signal

BeeAdHoc protocol discovers multiple paths to a destination. It then performs multipath routing to achieve load balancing and to reduce the path re-discovery time to cater for high node mobility scenarios. *BeeAdHoc* proportionally distributes foragers on multiple paths as a function of their quality; as a result, more foragers will follow the shortest cost path.

The forager DCs keep count of the number of foragers ($CountFgr_{dc}$) following different paths. After every UDINT period, *path use efficiency* for these paths is computed. The basic

motivation behind this definition is that the path use efficiency of shortest path ($PathUseEff_s$), relative to the path use efficiency of longest path ($PathUseEff_l$), would deteriorate once a malicious node tries to divert traffic on the larger and sub-optimal path. Each time the ratio ($\frac{PathUseEff_l}{PathUseEff_s}$) rises above the threshold ($FACTOR\_PATH$), the co-stimulation level for the longer path is raised. Finally, when the co-stimulation level for the longer path exceeds its threshold, the *danger signal* for that path is considered "HIGH". The forager DCs also have an inflammation flag – $FlagInfl_{path}$ – related to *path use efficiency* that is used to enhance the response to subsequently arriving relevant *non-self* Ags. These Ags are categorised as "dangerous" without any further co-stimulation if they keep arriving within the DC lifetime. Algorithm-1 describes the computation of forager path related *danger signal*.

6.3.3. Forager Route Info Related Danger Signal

Foragers also collect network state to estimate the quality of a traversed path. A node receiving foragers over a specific path would observe approximately the same average route information values. However, a malicious node may modify the route information carried in a forager. Therefore, if a node observes significant deviation in the average route information values of foragers, it may indicate the presence of *danger signal*.

In *iBeeAIS*, after every UDINT period, the forager DCs compute the average route information values carried by foragers on each path. To compute the average, the DCs use the values of $SumRtInfo_{dc}$ and $CountFgr_{dc}$, which they have collected during the last UDINT period.

$$Average\ RtInfo\ =\ SumRtInfo_{dc}\ /\ CountFgr_{dc} \tag{7}$$

Whenever average $RtInfo$ value of a path rises above (FACTOR_RTINFO × $AvgRtInfo_{dc}$) for the preceding UDINT period, the co-stimulation level for that path is raised. When co-stimulation level for a path exceeds the CO_STIMUL_RTINFO threshold, the *danger signal* for the path is considered to be "HIGH". Forager DCs also have an inflammation flag – $FlagInfl_{rtInfo}$ – related to the route information. The flag is used to categorise the subsequently arriving relevant *non-self* Ags as "dangerous" without the need for co-stimulation. Algorithm-1 describes the computation of *danger signal* related to forager route information.

6.4. B-Cells Processing

We use the set of collected DCs to tolerize and activate the set of scout and forager B-cells. The goal is to generate a set of pre-activated B-cells antibodies or detectors, which are capable of reacting only to the *non-self* Ags.

B-Cells Tolerization. The B-cells antibodies are matched with the Ags presented by semi-mature DCs, which represent the most recent system *self* state. The B-cells undergo negative selection to discard the *self* reactive lymphocytes, thus generating *self* tolerization. In BIS, *self* tolerization involves only the immature B-cells. Likewise, in *iBeeAIS* we exclude those B-cells from negative selection that are in the "activated" state.

Algorithm 1 : Danger signal computation

Require: Procedure to repeat every UDINT period of time for both scout and forager dendritic cells

```
//computation of scout danger signal
if (T_curr % UDINT = 0) then
  for (each DC_sct stored in DC_sct list) do
    if (FlagInfl_sct = TRUE) then
      decrement scout costimulation level
      if (scout costimulation level ≤ 0) then
        FlagInfl_sct = FALSE
        DC_sct state ← SEMIMATURE
      else
        DC_sct state ← MATURE
    compute average received FS_sd
    compute average received BS_ds
    if ((average recvd FS_sd > THRESH_RCVD_FS) || (average recvd BS_ds > THRESH_RCVD_BS)) then
      if (FlagInfl_sct = TRUE) then
        scout costimulation level ← CO_STIMUL_SCT
      else
        increment scout costimulation level
      if (scout costimulation level ≥ CO_STIMUL_SCT) then
        DC_sct state ← MATURE
        FlagInfl_sct = TRUE
    if (DC_sct state = IMMATURE) then
      DC_sct state ← SEMIMATURE

//computation of forager path related danger signal
if (T_curr % UDINT = 0) then
  for (each DC_fgr stored in DC_fgr list) do
    if (FlagInfl_path = TRUE) then
      DC_fgr state ← MATURE
  if (DC_fgr list size > 1) then
    compute PathUseEff for DC_fgr paths in DC_fgr list
    if (PathUseEff_l > FACTOR_PATH × PathUseEff_s) then
      pick DC_fgr with the longest RtLen in DC_fgr list
      if (FlagInfl_path = TRUE) then
        path costimulation level ← CO_STIMUL_PATH
      else
        increment path costimulation level for DC_fgr
      if (DC_fgr path costimulation level ≥ CO_STIMUL_PATH) then
        DC_fgr state ← MATURE
        DC_fgr FlagInfl_path ← TRUE

//computation of forager route info related danger
//signal
if (T_curr % UDINT = 0) then
  if (DC_fgr list size > 0) then
    for all (DC_fgr stored in DC_fgr list) do
      average RtInfo ← (SumRtInfo_dc / CountFgr_dc)
      if (FlagInfl_rtInfo = TRUE) then
        DC_fgr state ← MATURE
      if (average RtInfo > FACTOR_RTINFO × AvgRtInfo_dc) then
        if (FlagInfl_rtInfo = TRUE) then
          route info costimulation level ← CO_STIMUL_RTINFO
        else
          increment route info costimulation level
        if (route info costimulation level for this DC_fgr ≥ CO_STIMUL_RTINFO) then
          DC_fgr state ← MATURE
          FlagInfl_rtInfo ← TRUE
      else
        AvgRtInfo_dc ← average RtInfo
  for (each DC_fgr stored in DC_fgr list) do
    if (DC_fgr state = IMMATURE) then
      DC_fgr state ← SEMIMATURE
```

Algorithm 2 : B-cells Processing

Require: Procedure to repeat every UDINT period of time after scout/forager danger signal computation

```
//scout B-cells processing
if (T_curr % UDINT = 0) then
    for all (SEMIMATURE DC_sct stored in DC_sct list) do
        for all (BC_sct stored in BC_sct list) do
            if (FlagAct_sct = TRUE) then CONTINUE
            compute D_ham of DC_sct Ag and BC_sct antibody
            if (D_ham < EPS_NS_BC_SCT) then
                delete BC_sct
    while (BC_sct list size < NUM_BC_SCT) do
        generate a random BC_sct
        for all (SEMIMATURE DC_sct stored in DC_sct list) do
            compute D_ham of DC_sct Ag and BC_sct antibody
            if (D_ham < EPS_NS_BC_SCT) then
                delete BC_sct
            else
                push BC_sct to end of BC_sct list
    for all (MATURE DC_sct stored in DC_sct list) do
        for all (BC_sct stored in BC_sct list) do
            compute D_ham of DC_sct Ag and BC_sct antibody
            if (D_ham < EPS_NS_BC_SCT + ACT_FCTR_SCT) then
                if (DC_sct Ag exactly same as BC_sct antibody) then
                    FlagAct_sct ← TRUE
                    LifeAct_bc ← (T_curr + UDINT × rand() % 2 + 2)
                    BREAK
                if (T_curr < LifeAct_bc) then CONTINUE
                FlagAct_sct ← TRUE
                LifeAct_bc ← (T_curr + UDINT × rand() % 2 + 2)
                affinity_maturation (BC_sct antibody, SPEC_AB_SCT)
                BREAK

//forager B-cells processing
if (T_curr % UDINT = 0) then
    for all (SEMIMATURE DC_fgr stored in DC_fgr list) do
        for all (BC_fgr stored in BC_fgr list) do
            if (FlagAct_path || FlagAct_rtInfo) then CONTINUE
            compute D_ham of DC_fgr Ag and BC_fgr antibody
            if (D_ham < EPS_NS_BC_FGR) then delete BC_fgr
    while (BC_fgr list size < NUM_BC_FGR) do
        generate a random BC_fgr
        for all (SEMIMATURE DC_fgr stored in DC_fgr list) do
            compute D_ham of DC_fgr Ag and random BC_fgr antibody
            if (D_ham < EPS_NS_BC_FGR) then
                delete BC_fgr
            else push BC_fgr to end of BC_fgr list
            endif
    for all (MATURE DC_fgr stored in DC_fgr list) do
        for all (BC_fgr stored in BC_fgr list) do
            compute D_ham of DC_fgr Ag and BC_fgr antibody
            if (D_ham < EPS_NS_BC_FGR + ACT_FCTR_FGR) then
                if (DC_fgr Ag exactly same as BC_fgr antibody) then
                    if (DC_fgr FlagInfl_path = TRUE) then
                        FlagAct_path ← TRUE
                    if (DC_fgr FlagInfl_rtInfo = TRUE) then
                        FlagAct_rtInfo ← TRUE
                        AvgRtInfo_bc ← AvgRtInfo_dc
                    LifeAct_bc ← (T_curr + UDINT × rand() % 2 + 2)
                    BREAK
                if (T_curr < LifeAct_bc) then CONTINUE
                if (DC_fgr FlagInfl_path = TRUE) then
                    FlagAct_path ← TRUE
                if (DC_fgr FlagInfl_rtInfo = TRUE) then
                    FlagAct_rtInfo ← TRUE
                    AvgRtInfo_bc ← AvgRtInfo_dc
                LifeAct_bc ← (T_curr + UDINT × rand() % 2 + 2)
                affinity_maturation (BC_fgr antibody, SPEC_AB_FGR)
                BREAK
```

B-Cells Activation. The B-cells that survive *self* tolerization, undergo an activation phase by matching with the *non-self* Ags presented by mature DCs. The activation flag of a B-cell is raised if its antibody matches a *non-self* Ag with affinity higher than the B-cells *Activation Factor* (ACT_FCTR_SCT, ACT_FCTR_FGR). We define the B-cells activation factor as, *"a measure of how distant from the non-self Ags the detectors may be, to be selected for activation"*. The value of activation factor affects the number of possible B-cells that might be activated.

B-Cells Affinity Maturation. The activated B-cells undergo affinity maturation to enable a more specific response against the *non-self* Ag. We carry out mutation of B-cells antibodies to achieve the desired *Antibody Specificity* (SPEC_AB_SCT, SPEC_AB_FGR), which we define as, *"the measure of how close the antibody needs to be to the non-self Ag"*. The result of processing the B-cells is a set of detectors tolerant to *self*, with few possibly activated against *non-self* Ags. Algorithm-2 describes the process for scout and forager BCs.

6.5. Elimination or De-Activation of Immune Components

After every UDINT period of time the DCs that complete their lives during the preceding UDINT period are eliminated from the system. The surviving DCs are then refreshed to restart the process of Ag sampling and determination of the *danger signals*. In this context, the data gathering fields of the scout DCs ($Count_{FS}$, $Count_{BS}$) and forager DCs ($CountFgr_{dc}$, $SumRtInfo_{dc}$) are reset. The state of the surviving *semi-mature* or *mature* DCs is also changed to *immature*.

In BIS, the activated B-cells after some time revert back to their steady states. Similarly, in *iBeeAIS*, at the end of every UDINT time period, B-cells that have their activation flags "TRUE" but active life expired, are de-activated, i.e $FlagAct_{sct}$, $FlagAct_{path}$ and $FlagAct_{rtInfo}$ are set to "FALSE". The average route information field in de-activated forager B-cells is also reset to zero.

6.6. Matching Antigens with B-Cells

During operation, the system matches scout or forager Ags with their respective B-cells to classify them as *self* or *non-self* (Algorithm-3). In case of match with an activated B-cell, a *non-self* Ag is identified and corresponding appropriate action is initiated. For matching we use a parameter *Detection Range* (DET_RANGE_SCT, DET_RANGE_FGR) for B-cell antibodies, which we define as *"a measure of how far away from itself a detector can detect Ags"*. Detection range of a *non-self* Ag must be at least equal to the antibody specificity for successful detection. In case of a successful match, the identified *non-self* Ag is dropped. However, under modified route information attack when $FlagAct_{rtInfo}$ is found raised, route information in forager's header is replaced with the expected value computed as:

$$forager\ RtInfo = T_{curr} - AvgRtInfo_{bc} \tag{8}$$

Algorithm 3 : Matching Ags with B-cells

Require: Procedure to execute on receiving every *scout* or $forager$ after *DC* formation

```
//matching scout Ags with scout B-cells
```

for all (FS and BS received at each $node_i$) **do**
  for all (BC_{sct} stored in BC_{sct} list) **do**
    compute D_{ham} of scout Ag and BC_{sct} antibody
    if ((D_{ham} < DET_RANGE_SCT) && ($FlagAct_{sct}$ = TRUE)) **then**
      scout declared Ag_{ns}, dropped
      $LifeAct_{bc} \leftarrow (T_{curr}$ + UDINT × $rand()$ % 2 + 2)

```
//matching forager Ags with forager B-cells
```

for all (foragers received at each $node_i$) **do**
  for all (BC_{fgr} stored in BC_{fgr} list) **do**
    compute D_{ham} of forager Ag and BC_{fgr} antibody
    if ((D_{ham} < DET_RANGE_FGR) && ($FlagAct_{rtInfo}$ = TRUE)) **then**
      forager declared Ag_{ns}
      forager $RtInfo \leftarrow T_{curr}$ - $AvgRtInfo_{bc}$
      $LifeAct_{bc} \leftarrow (T_{curr}$ + UDINT × $rand()$ % 2 + 2)
      BREAK
  for all (BC_{fgr} stored in BC_{fgr} list) **do**
    compute D_{ham} of forager Ag and BC_{fgr} antibody
    if ((D_{ham} < DET_RANGE_FGR) && ($FlagAct_{path}$ = TRUE)) **then**
      $LifeAct_{bc} \leftarrow (T_{curr}$ + UDINT × $rand()$ % 2 + 2)
      forager declared Ag_{ns}, dropped
      BREAK

7. *iBeeAIS* Attack Simulations

To validate *iBeeAIS* security, we implemented the system in ns-2 and launched a number of routing attacks. The efficacy of *iBeeAIS* is evaluated under three scenarios: (1) *Normal Routing* in which *iBeeAIS* protocol is evaluated with no malicious nodes in the network, (2) *Partially Functional Under Attack* where *iBeeAIS* is in secure mode but malicious packets are not dropped, which makes the attacks successful, and (3) *Fully Functional Under Attack* where *iBeeAIS* drops detected scouts/foragers and replaces forged routing information with expected values.

7.1. Node Topology

We use the 9 node topology of Fig. 6 that we have first used for *BeeSec* [25]. It is a rectangular area of $1000 \times 500\, m^2$, where *node 0* is the source and *node 8* is the destination. Three distinct paths exist between the source and the destination nodes – *0-7-8*, *0-5-6-8* and *0-1-2-3-4-8*. The path *0-7-8* is the shortest one and is discovered first. Therefore, under no-attack conditions, majority of foragers follow this path. In contrast, virtually no foragers follow the path *0-1-2-3-4-8* under normal routing conditions because it is the least suboptimal path.

7.2. Routing Attacks

We use an *attacker framework* implemented in ns-2 to launch four types of scout and forager related attacks on *iBeeAIS*. In each attack scenario, we monitor the routed traffic at three points in the network – *node 2*, *node 5* and *node 7* – and then generate traffic maps to indicate the success or failure of the attack. We now discuss the details of each attack.

Attack-1: Forging Forward Scout. This attack is launched 100 seconds after the start of simulation, when initial route discovery is complete. The attacker *node 4* launches fake forward scouts to install a forged route *0-1-2-3-4-8*. The attack rate is approximately 12 to 13 fake scouts per second. The fake packets have *node 0* as source and *node 8* as destination.

Attack-2: Forging Backward Scout. The attack involving spoofed backward scouts is launched by *Node 2* at time t=100 seconds. Forged backward scouts are generated at the rate of approximately 11 to 12 scouts per second with D_{sct} as *node 8*.

Attack-3: Forging Spoofed Forager. At t=50 seconds, the attacker *node 5* sends forged foragers to install a forged path *0-1-2-3-4-8* at *node 0*. The attack packet rate is approximately 4 foragers per second. The routing information is also modified in forged packets; delay value carried in packet header is artificially reduced to misrepresent the shortest path.

Attack-4: Modifying Forager Route Information. In this attack, the malicious *node 7* artificially increases the route delay values in the foragers returning from *node 8* to *node 0*; thus making the path *0-7-8* undesirable. The attack is launched at simulation time t=100 seconds.

The attack simulations show that in the absence of malicious nodes, *iBeeAIS* protocol routes maximum data packets over the shortest path *0-7-8*, and almost no data packets through the least optimum path *0-1-2-3-4-8*. However, attacks on *iBeeAIS* are successful when protocol is running with partial functionality; *non-self* Ags continue their journey to their destinations and cause fake routes to be established. As a result, the network traffic is diverted to an attacker's desired path. Now when *iBeeAIS* is running with full functionality, our system successfully detects the *non-self* Ags and drops them to preserve the *BeeAdHoc* normal routing behavior. To conserve space we do not show here the *iBeeAIS* traffic maps that demonstrate the effect of these routing attacks. Interested readers are referred to the Technical Report [36] that has the traffic plots generated during attacks on *iBeeAIS* protocol.

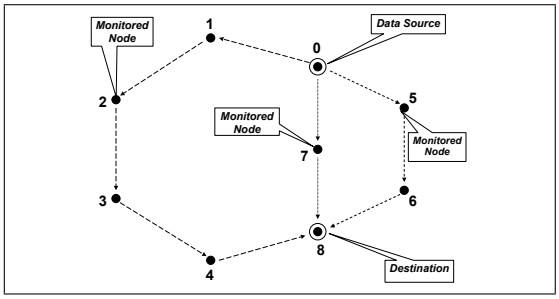


Figure 6: Node topology selected for attacks

7.3. Detection Performance

We determine the *iBeeAIS* detection performance by launching the above mentioned attacks and varying the transmission rates of data and attack packets. Recall that when an attack is launched *iBeeAIS* can detect it after a certain delay, which will

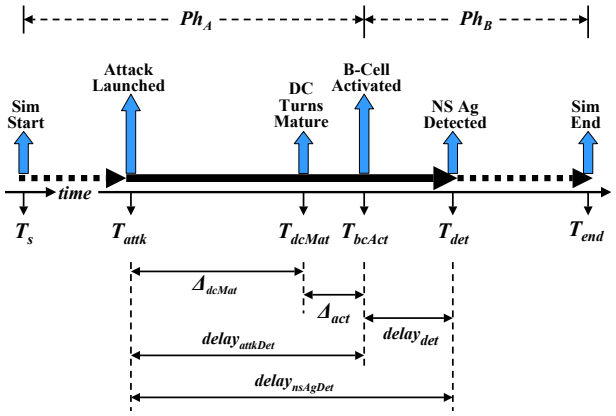


Figure 7: Delays involved in *iBeeAIS* detection process

be multiples of UDINT. We define three types of delays in the detection process, starting from the attack's launch time. Refer to Fig. 7 for a better understanding.

$\boldsymbol{\Delta_{dcMat}}$: The number of UDINT time periods taken by the first DC to do a transition to the mature state. This consists of the time intervals needed for detection of *danger signal* and the wait period for co-stimulation. It indicates how quickly the system can sense "danger".

$\boldsymbol{delay_{attkDet}}$: The time delay from the attack's launch time to its detection time. This delay equals to $(\Delta_{dcMat} + \Delta_{act}) \times$ UDINT, where Δ_{act} is the number of UDINT periods taken by a mature DC Ag to activate the first B-cell. Δ_{act} is a measure of the adequacy of B-cells coverage. A lower value for Δ_{act} indicates better detector coverage and ideally it should be zero; a mature DC Ag should immediately match a de-activated B-cell antibody.

$\boldsymbol{delay_{nsAgDet}}$: The overall time taken by the system to detect the first *non-self* Ag. It equals to $(delay_{attkDet} + delay_{det})$, where $delay_{det}$ is the time taken by an activated B-cell to match an incoming *non-self* Ag.

In *iBeeAIS*, we divide the attack simulation into two phases to measure detection rates:

- $\boldsymbol{Ph_A}$: The pre-activation phase, from the start of simulation (T_s) to the first activation of a B-cell (T_{bcAct}).
- $\boldsymbol{Ph_B}$: The post-activation phase, which is the remaining simulation period after the B-cells activation (T_{bcAct}).

We measure *DR* and *FAR* [2] for all the four attacks. Note that because of static network structure, no scouts are launched after initial route discovery. As a result, only malicious scouts traverse the network under attack scenarios. Therefore, in scout attacks, *FP*, *TN* and *FAR* make little sense.

Detection Rates. We have tabulated our Ag detection results in Table 3. In Attack-1, all *non-self* forward scout Ags launched by malicious *node 4* are detected; as a result, DR is 100% during the post-activation phase (Ph_B). Ag detection is mostly done by the two nodes neighbouring the attacker – *node 3* and *node 8*. It is interesting to note that the average *DR* is reduced to 96.26%. This is because on the average 3.74% of the total *non-self* Ags have arrived at the node in the pre-activation phase (Ph_A), when the system has just started learning about the attack. These *non-self* Ags are mis-classified as *self*. Similarly, in Attack-2, Attack-3 and Attack-4, the detecting nodes – *node 1* and *node 0* – that receive the forged bee agents from the attacker nodes – *node 2*, *node 5* and *node 7* – achieve a detection rate of 100% in Ph_B because they successfully detect all *non-self* Ags.

However, the protocol is unable to detect 3.53%, 7.03% and 2.63% of the *non-self* Ags that have arrived during Ph_A. This has resulted in lowering the average *DRs* of Attack-2, Attack-3 and Attack-4 to 96.47%, 92.97% and 97.37%, respectively. Remember that in forager related attacks, the detecting nodes have to perform simultaneous detection of both *self* and *non-self* Ags; therefore, we can calculate the *FAR* of these attacks. In Attack-4, none of the *self* Ags are incorrectly classified as *non-self*, which results in *FAR* of 0%. In comparison, in Attack-3 we have a *FAR* of just 0.702%.

Table 3: *iBeeAIS* Ag detection rates

Attack-1: Forging Forward Scout

Phase	*Ags Recvd*	*Average Ags Detected*				*FAR (%)*	*DR (%)*
		FP	*TN*	*TP*	*FN*		
Ph_A	338.8	-	-	0	338.8	-	0
Ph_B	8729.8	-	-	8729.8	0	-	100
Total	9068.6	-	-	8729.8	338.8	-	96.26

Attack-2: Forging Backward Scout

Phase	*Ags Recvd*	*Average Ags Detected*				*FAR (%)*	*DR (%)*
		FP	*TN*	*TP*	*FN*		
Ph_A	291.6	-	-	0	291.6	-	0.0
Ph_B	7970	-	-	7970	0	-	100.0
Total	8261.6	-	-	7970	291.6	-	96.47

Attack-3: Forging Spoofed Forager

Phase	*Ags Recvd*	*Average Ags Detected*				*FAR (%)*	*DR (%)*
		FP	*TN*	*TP*	*FN*		
Ph_A	1126.2	0	985.8	0	140.4	0.0	0.0
Ph_B	6340	38.4	4445.2	1856.4	0	0.856	100.0
Total	7466.2	38.4	5431	1856.4	140.4	0.702	92.97

Attack-4: Modifying Forager Route Information

Phase	*Ags Recvd*	*Average Ags Detected*				*FAR (%)*	*DR (%)*
		FP	*TN*	*TP*	*FN*		
Ph_A	1794.4	0	1658	0	136.4	0.0	0.0
Ph_B	6805.6	0	1760.4	5045.2	0	0.0	100.0
Total	8600	0	3418.4	5045.2	136.4	0.0	97.37

Detection Delays. The *non-self* Ag detection delays are tabulated in Table 4. The two scout related attacks have similar values for $delay_{attkDet}$ and the $delay_{nsAgDet}$. This is because of the fact that the *iBeeAIS* has same DC processing algorithm for both scout related attacks and the system processes the incoming scout Ags in the same way. Therefore, the detection performance for both types of scout related attacks is similar. How-

[2] $DR = TP/(TP+FN)$ and $FAR = FP/(FP+TN)$, where TP=true positive, TN=true negative, FP=false positive and FN=false negative.

Table 4: *iBeeAIS* Ag detection delays

Average Number of UDINT periods		*Average Delay Values*		
Δ_{dcMat}	Δ_{act}	$delay_{attkDet}$ *(sec)*	$delay_{det}$ *(ms)*	$delay_{nsAgDet}$ *(sec)*
Attack-1: Forging Forward Scout				
3	0	15	12.6	15.0126
Attack-2: Forging Backward Scout				
3	0	15	35.8	15.0358
Attack-3: Forging Spoofed Forager				
6.8	0	34	95.22	34.09522
Attack-4: Modifying Forager Route Information				
2	0	10	48.0	10.048

ever, the detection delays for forager related attacks – Attack-3 and Attack-4 – are significantly different. This is because different mechanisms for DC differentiation are employed by *iBeeAIS* for these two types of attacks: forging source route and forging route information. The forager path related DC transition to maturity depends on the relative rates of foragers on different paths. The forager rates on different paths are highly sensitive to topology because of node mobility.

We also see in Table 3 and Table 4 that a lower aggregate *DR* corresponds to a higher detection delay and vice versa. Attack-3 has relatively higher detection delay of 34.095 *seconds*; consequently it has a relatively lower aggregate *DR* of 92.97% compared with other attacks. This implies that we need to reduce the B-cells activation time in order to minimize the number of *non-self* Ag that arrive in Ph_A. As a consequence, it will implicitly improve *DR*.

CPU Cycles for Detection. Table 4 shows that the Δ_{act} value for each of the four attacks is zero, i.e a B-cell gets activated in the same UDINT period in which a DC turns mature. We also measured the CPU cycles taken by a mature DC Ag to activate the first B-cell. This will provide us insight about the delays during the updating of detector sets. Our results are shown in Table 5. The values are averaged over five independent runs for each of the four attacks. The results show that a mature DC takes relatively more CPU cycles in activating the B-cells when it updates the detectors set for the first time. An important reason is that the B-cells activation for the first time also involves affinity maturation of antibodies – a step not needed for subsequent detector set updates. It just takes 4.61% to 29.22% of the initial CPU cycles for the subsequent processing/activation of the same B-cells in all four attacks. We also convert the computed CPU cycles to show time, based on clock speed of the processor (1.8 MHz). If we analyze time in Table 5, we can conclude that our B-cell activation algorithm takes negligible amount of time compared with the overall activation delay of $delay_{attkDet}$ since the launch of attack.

Moreover, we also measure the CPU cycles – within the time interval of $delay_{det}$ – taken by an activated B-cell to match a newly arrived *non-self* Ag. We can see in Table 5 that our system takes just 3.74 μsec to 10.3 μsec to match an activated B-cell with the corresponding *non-self* Ag. The delay is negligible compared with the time to activate a B-cell for the first time.

8. Comparing *iBeeAIS* with Self/Non-self Model, Danger Theory and Conventional Approaches

In this section, we compare the performance of our *iLite* security framework with existing AIS and conventional security approaches. We have selected two well known AIS based approaches from AIS literature: (1) *self/non-self* discrimination, and (2) *danger theory*. We also compare our system with a conventional security system – utilizing asymmetric cryptography based digital signatures – for protocol security. In order to do a fair comparison, we do empirical investigations of a security system along three dimensions: (1) its ability to do well in case of non changing *self*, (2) its ability to adaptively learn the changing *non-self*, (3) its ability to adaptively learn the changing *self*. The first step is to secure the same *BeeAdHoc* protocol with each of the above-mentioned security approaches. *BeeAIS* [38] utilizes *self/non-self* discrimination, *BeeAIS-DC* [40] uses *danger theory*, and *BeeSec* [25] provides protection using cryptography. We have implemented all these systems in ns-2 in order to subject them to the same attack scenarios. This will enable reliable and unbiased analysis about the relative merits/demerits of each approach. In order to make the paper self contained, we summarize the key characteristics of each security protocol.

BeeAIS. It is a negative selection based AIS security framework for *BeeAdHoc*. It first learns the system *self* during an initial *learning phase* of 50 seconds and then monitors the system to detect *non-self* associated with the malicious activity. *BeeAIS* utilizes three types of antigens: (1) *scout antigen*, (2) two *forager antigens* (Type-I and Type-II). The scout antigen is designed to detect anomalies in the forward and backward scouts. Similarly, the two forager antigens detect tampering of source route and the routing information carried by a forager.

BeeAIS-DC. It detects routing misbehavior using the concepts from danger theory. It utilizes an extended version of the basic DC model of DCA (proposed in [49]) for MANETs. The extended model is discussed in Section 4.1. Remember original *BeeAIS-DC* utilizes scout antigens/detectors to detect only scout related attacks on *BeeAdHoc*. The dendritic cells are used to sense danger and update the scout detector sets to adapt to

Table 5: Average CPU cycles and time taken for activation of B-cells and matching the *non-self* Ags

Mature DC Activating a B-cell by Updating Detector Sets					*Activated B-cell detecting Non-Self Ag*	
initial values		*subsequent values*		*% of initial cycles*		
CPU cycles	*Time μ sec*	*CPU cycles*	*Time μ sec*		*CPU cycles*	*Time μ sec*
Attack-1: Forging Forward Scout						
62829.6	34.9	2894.5	1.61	4.61%	7139.4	3.96
Attack-2: Forging Backward Scout						
65619	36.5	6399.4	3.56	9.75%	9150.8	5.08
Attack-3: Forging Spoofed Forager						
148720	82.6	43462.5	24.1	29.22%	18584	10.3
Attack-4: Modifying Forager Route Information						
133020	73.9	34032.6	18.9	25.58%	6727.3	3.74

changing system *self*. For a fair comparison, we have enhanced *BeeAIS-DC* with forager antigens/detectors to empower it to detect forager related attacks as well.

BeeSec. It is a digital signature based security framework to protect *BeeAdHoc* against malicious scouts and foragers using public key cryptography. *BeeSec* uses digital signatures to perform *packet authentication* to ensure that the data carried in the header fields of scouts and foragers – source address, destination address, packet ID, routing information – has been sent by authorized nodes. The *integrity check* of the source route is also done to ensure that no valid node on the route has been removed by a malicious node. Consequently, tampering and fabrication attacks in *BeeAdHoc* are prevented by *BeeSec*.

Table 6: *BeeAIS* Ag detection rates

Ags Recvd		*Average Ags Detected*				*FAR*	*DR*
total	*non-self*	*FP*	*TN*	*TP*	*FN*	*(%)*	*(%)*
Attack-1: Forging Forward Scout							
2961	2961	-	-	2941	20	-	99.33
Attack-2: Forging Backward Scout							
3987	3987	-	-	3987	0	-	100.00
Attack-3: Forging Spoofed Forager							
29016	309	0	28707	309	0	0.00	100.00
Attack-4: Modifying Forager Route Information							
19642	5450	9	14183	3722	1728	0.0634	68.29

Table 7: *BeeAIS-DC* Ag detection rates

Ags Recvd		*Average Ags Detected*				*FAR*	*DR*
total	*non-self*	*FP*	*TN*	*TP*	*FN*	*(%)*	*(%)*
Attack-1: Forging Forward Scout							
9691	9691	-	-	8375	1316	-	86.42
Attack-2: Forging Backward Scout							
6522	6522	-	-	6058	464	-	92.88
Attack-3: Forging Spoofed Forager							
33934	389	58	33487	0	389	0.173	0.00
Attack-4: Modifying Forager Route Information							
29819	8355	0	21464	4670	3685	0.00	55.89

8.1. Case 1: The Ability to Do Well in Case of Non-Changing Self

In this case, our objective is to develop an insight about the ability of different systems to detect previously unseen malicious behavior. Therefore, the same routing attacks, to which *iBeeAIS* in Section 7 is subjected, are launched on *BeeAIS*, *BeeAIS-DC* and *BeeSec*. (We are using the same topology as well.) We now discuss the outcome of our study. In this case, we use *DR* and *FAR* metrics for comparison.

8.1.1. Comparison with Self/Non-self Based BeeAIS

The antigen detection results of *BeeAIS* for the four routing attacks are tabulated in Table 6. The *BeeAIS* protocol, like *iBeeAIS*, is able to prevent these attacks by detecting and dropping the respective *non-self* antigens. These attacks constitute *new malicious behavior* which is experienced by the nodes as explained in Section 1. The attacks are detected based on the difference from an initial learnt notion of the system's *self*. If we compare this detection performance with that of *iBeeAIS* from Table 3, we see that *iBeeAIS* provides consistent and superior performance compared with *BeeAIS*. (Note that one has to focus on the performance of *i*BeeAIS in Ph_B because by then it has detected the danger signals and activated the B-cells.)

8.1.2. Comparison with Danger Theory Based BeeAIS-DC

Table 7 shows the antigen detection results of *BeeAIS-DC*. It is evident that *BeeAIS-DC* is able to counter the scout related attacks with a good detection rate; however, forager related attacks are not prevented. This is because *BeeAIS-DC* is unable to detect *non-self* forager antigens by using activated T-cells that cannot improve their affinity with the *non-self* antigens to achieve a better match. The same attacks are detected by *iBeeAIS*, Table 3 because *iBeeAIS* uses activation of B-cells, which undergo affinity maturation for a more focused response against suspected *non-self* antigens.

To conclude, both *BeeAIS* and *iBeeAIS* (as expected) are able to detect previously unseen attacks albeit *iBeeAIS* has relatively better performance in all attack scenarios. In comparison, *BeeAIS-DC* is unable to detect forager related attacks.

8.2. Case 2: The Ability to Adaptively Learn the Changing Non-self

It is important to understand that a *non-self* can change when a *benign behavior turns malicious*. This is possible when a malicious node tries to install a previously valid route that is not optimal. In this section, we analyze the ability of different AIS approaches to detect the changing *non-self*. This is done by launching a new scout routing attack on the protocols. We now describe the attack.

Attack-5: Returning Scouts with a Suboptimal Route. The attack is launched by *Node 5* (see Figure 6) at time $t = 1$ *sec*, when the CBR traffic source starts sending data to the routing layer. (*Node 5* is greedy and does not want to deplete its battery for routing packets.) Once the node receives a forward scout, it immediately unicasts a backward scout with the path *8-4-3-2-1-0* in its header. Since this scout will arrive first, it will trick the *Node 0* to use it as an optimal path. As a result, the majority of foragers follow *0-1-2-3-4-8* instead of *0-7-8* and *0-5-6-8*.

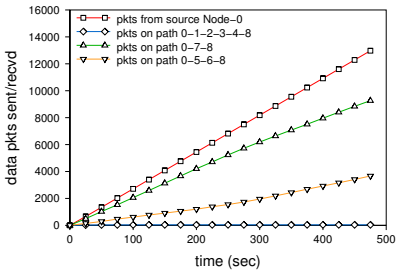
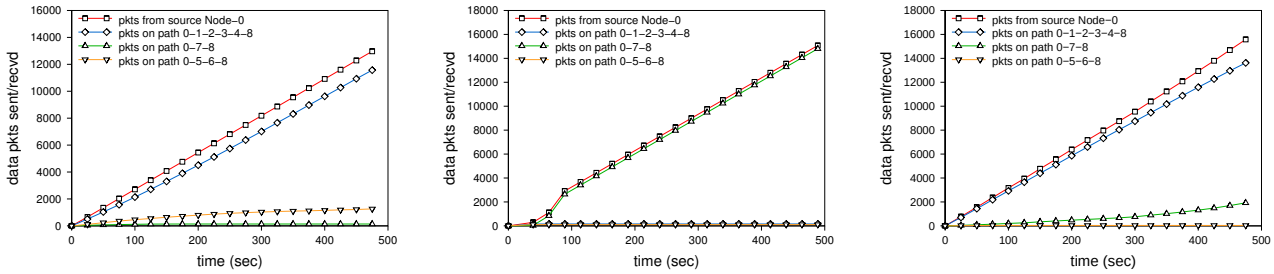


Figure 8: BeeAIS, BeeAIS-DC & iBeeAIS normal routing



(a) BeeAIS routing under attack (b) iBeeAIS routing under attack (c) BeeAIS-DC routing under attack

Figure 9: Attack-5 on BeeAIS and iBeeAIS

8.2.1. Comparison with Self/Non-self Based BeeAIS

It is important to mention that since the attack is launched in the beginning of the *learning phase* of *BeeAIS* ; therefore, it becomes part of its *self* and hence *BeeAIS* would not be able to detect it. In this case, it makes little sense to compute *DR* and *FAR* because *BeeAIS* does not treat the malicious agents as tampered. Therefore, we use traffic plots to show that the attack has been successfully launched (compare traffic patterns in Figures 8 and 9(a)) and *BeeAIS* is unable to counter it. In comparison, the *iBeeAIS* protocol, detects and prevents this attack as shown in Figure 9(b). The integrated AIS detection process has the mechanism to learn the changing *non-self* through feedback from DCs (indicating damage done to the network); as a result, new behavior can be classified as as *self* or *non-self*. Initially, the traffic was routed on suboptimal path but this abnormality is detected by forager DCs (that sense danger), which indicates the presence of *non-self* antigens. Consequently, *iBeeAIS* successfully adapts itself to identify *benign turns malicious* behavior as malicious, and detects/drops the *non-self* antigens to ensure that traffic is routed on the optimal path.

8.2.2. Comparison with Danger Theory Based BeeAIS-DC

We also launched Attack-5 on *BeeAIS-DC* but it was unable to detect it as shown by the traffic graph of Figure 9(c). It is interesting to note that *BeeAIS-DC* protocol was able to sense the danger after two UDINT periods (at $t = 10$ *secs*) when packets were being routed over suboptimal route *0-1-2-3-4-8*. However, it is unable to discriminate between *self* and *non-self* forager antigens because it does not use affinity maturation (done by *iBeeAIS*) to tune its detectors to achieve a better match with the suspected *non-self* antigens. As a result, *BeeAIS-DC* considers the *non-self* antigens also as *self*.

8.3. Case 3: The Ability to Adaptively Learn the Changing Self

Once we incorporate mobility in MANETs, new valid routes might emerge and invalid (or unavailable) routes might become valid (or available). This behavior represents *new benign* and *malicious turns benign* situation that results because of changing *self*. Therefore, it is important to learn changing *self* under normal mobility scenarios. We use *FAR* as a measure to show the effectiveness of this ability. Ideally, a system should have a 0% FAR under normal mobility conditions, i.e, the system should correctly identify the new/changed *self* as *self* and not as *non-self*.

Mobility Simulations Scenario. We use a rectangular area of 2400×480 m^2. The nodes move according to the "random waypoint" model: a node's speed is selected from a uniform distribution ranging from 1 m/s to 20 m/s and the pause time is in between 1 to 20 seconds. Our test scenarios are derived from the base scenario in [39], where each node acts as a source and destination at the same time. Nodes are formed into pairs and a given node can send/receive data only from its pair. The pair forming uses a sequential algorithm: first node is the pair of the last node, second of the second last and so on. A node generates about 30 packets/second of constant bit rate (CBR) data with fixed packet size of 512 bytes. The results are reported for network sizes of 10, 30 and 50 nodes.

Comparison of Protocol FARs. FARs of *BeeAIS*, *BeeAIS-DC* and *iBeeAIS* are tabulated in Table 8. As expected, *BeeAIS* has a significantly high FAR for scout antigens, which shows

Table 8: Comparison of FAR of different Protocols under normal mobility scenarios

Nodes	*Protocol*	*Average Ags Processed*			*FAR*
		Type	*Count*	*FP*	*(%)*
10	*BeeAIS*	*scout*	586.4	395.0	67.360
		Type-I fgr	30654.8	4.8	0.015
		Type-II fgr	30650.0	318.6	1.039
	BeeAIS-DC	*scout*	1218.2	0.0	0.000
		forager	101005.6	3.2	0.00317
	iBeeAIS	*scout*	1211.2	0.0	0.000
		forager	100404.2	2.6	0.00259
30	*BeeAIS*	*scout*	8297.0	2542.2	30.639
		Type-I fgr	58911.2	38.0	0.064
		Type-II fgr	58873.2	1099.8	1.868
	BeeAIS-DC	*scout*	14325.8	0.0	0.000
		forager	145464.2	10.4	0.00715
	iBeeAIS	*scout*	14097.8	0.0	0.000
		forager	143868.8	13.0	0.00904
50	*BeeAIS*	*scout*	14106.0	2247.6	15.933
		Type-I fgr	81798.8	51.2	0.062
		Type-II fgr	81747.6	3034.0	3.711
	BeeAIS-DC	*scout*	39336.2	0.0	0.000
		forager	149294.8	1.4	0.00094
	iBeeAIS	*scout*	41214.6	0.0	0.000
		forager	151930.6	0.8	0.00053

Table 9: Processing Overhead of BeeSec Protocol

Nodes	*Avg Agents*			*CPU Cycles*	*Cycles per*
	type	*Tx/Rx*	*count*	*consumed*	*Agent*
10	***FSct***	Tx	512	3537916374	30307814
		Rx	508	11886093722	
	BSct	Tx	39	77086991	7744092
		Rx	31	178792612	
	Fgr	Tx	24154	165985621959	17326347
		Rx	23813	248950044446	
30	***FSct***	Tx	3738	26013787491	45547781
		Rx	13371	515966870191	
	BSct	Tx	794	1294496810	5042711
		Rx	628	2142964238	
	Fgr	Tx	80440	630642104467	22719089
		Rx	79376	1181050073853	
50	***FSct***	Tx	9869	81941385831	61823206
		Rx	53400	2857984054604	
	BSct	Tx	2035	2816227557	3579635
		Rx	1602	3517576016	
	Fgr	Tx	68326	422443128120	17954918
		Rx	67089	789782488318	

Table 10: Processing Overhead of *i*BeeAIS Protocol

Nodes	*Avg Agents*		*Avg Cycles*		*Cycles per*
	type	*count*	*agent proc*	*DC proc*	*Agent*
10	***Sct***	834	20708030	22934667	52329
	Fgr	25492	696024392	10222580	27704
30	***Sct***	14156	474235285	549279148	72302
	Fgr	80678	2502929090	87293193	32105
50	***Sct***	53542	4001982149	1863150030	109542
	Fgr	87208	2816204692	140654500	33906

that it is unable to learn the changing *self*; the new scouts launched to discover new routes under mobility scenario are treated as malicious and dropped. Consequently, due to non-availability of routes, *BeeAIS* transports smallest number of foragers at destinations (please note that *Type-II foragers* are a subset of *Type-I foragers*); *iBeeAIS* transports from 85.7% to 227.5% more foragers than *BeeAIS* for different size networks. Since new routes are not discovered; therefore, new foragers for these routes are not launched and the FAR of foragers remains relatively small. In comparison, *BeeAIS-DC* and *iBeeAIS* are able to learn the changing *self* by taking feedback from DCs; as a result, both of them have low FAR.

8.4. Comparison with Cryptography Based BeeSec

The operation of *BeeSec* security system does not depend upon behavioral learning and is, therefore, invariant to any changes in the system *self* or *non-self*. *BeeSec* identifies malicious packets by authenticating scouts and forgers by using digital signatures. *BeeSec* is, therefore, able to detect all forged/tampered scouts and foragers to achieve 100% DR with 0% FAR. It is also able to detect Attack-5 because *Node 5* spoofs the ID of *Node 1* and hence digital signatures do not match. To conclude, *BeeSec* provides best security among all approaches; however, as mentioned in [25] its very large processing and communication overheads make it infeasible for deployment on battery constrained mobile nodes. We now compare its processing and communication overheads with the best AIS approach: *iBeeAIS*.

Table 11: Communication Overhead of BeeSec and *i*BeeAIS

Nodes	*Average Control Packet Bytes*		*Percent Rise for*
	iBeeAIS (kbyte)	*BeeSec (kbyte)*	*BeeSec*
10	8,21,787	92,38,361	1024.18%
30	14,54,723	149,22,953	925.83%
50	18,90,321	177,51,907	839.09%

8.4.1. Processing and Communication Overheads

Remember we measure processing overhead in terms of cycles of a processor needed to process a scout and forager. Similarly, communication overhead is total number of bytes of control packets transferred to discover and maintain valid routes.

The processing overheads of *BeeSec* and *iBeeAIS* are shown in Tables 9 and 10 respectively. In *BeeSec*, CPU cycles are measured when the protocol performs cryptographic functions on receiving or transmitting an agent. In *iBeeAIS*, the agent processing time also includes the time of handling DCs, B-cells and matching antigens. Therefore, CPU cycles for all phases of *i*BeeAIS processing are logged. We report average processing cycles for scout and foragers. It is clear from Tables 9 and 10 that the agent processing time of *iBeeAIS* is significantly small compared with that of *BeeSec*. *BeeSec* takes from 52854.99% to 72616.67% (depending on the network size) more CPU cycles compared with *i*BeeAIS.

The communication overheads of *BeeSec* and *iBeeAIS* are tabulated in Table 11. It is obvious from Table 11 that the communication overhead of *BeeSec* is an order of magnitude greater than that of *iBeeAIS*. *BeeSec* transmits from 839% to 1024% (depending on the network size) more control bytes compared with *i*BeeAIS. This analysis shows the benefits of lightweight enhancements adopted in *iBeeAIS*; as a result, it provides similar security compared with *BeeSec* but with very small processing and communication costs. This concludes the comparative study of our system with different AIS based techniques and a conventional system. Now we focus our attention on the network performance of different protocols.

9. Network Performance

We evaluated the network performance of our proposed *iBeeAIS* protocol, under normal routing without attacks, through extensive simulations in the ns-2 simulator. We have compared the integrated AIS security framework with five protocols: (1) *BeeAdHoc*, (2) a cryptographic security system, *BeeSec*, (3) *self/non-self* based system *BeeAIS*, (4) non-secure *DSR* and (5) non-secure *AODV*.

9.1. Simulation Topology

The network performance of protocols is determined using the same ns-2 simulation scenario as described in Section 8.3. For these simulations, we gradually scale the size of network from 10 nodes to 150 nodes. We, however, cannot report results for *BeeSec* beyond 60 nodes network because ns-2 is unable to execute *BeeSec* due to its high resource requirements.

For each of the network scenarios, we do twenty repetitions of randomly seeded simulation runs, each lasting 1000 seconds. The results are averaged to factor out stochastic elements in the algorithm/environment. The reported results satisfy the 90% confidence level. The performance parameters used to compare the protocols, adopted from [31], include:

Average throughput. *The total number of data bits delivered to destination nodes during the simulation divided by the total simulation time.*

Transmission efficiency. *The number of data bytes delivered to the application layer at destination nodes at the cost of a unit control byte.*

Control overhead. *The total number of control bytes (Mbytes) – including headers of both control packets and data packets – transmitted by all nodes in the network.*

Energy efficiency. *The total energy consumed, for both data and control traffic, per 1000 bits of data delivered to the destination (Joules/kbits).*

9.2. Protocol Network Performance

The results of our network performance simulations are shown in Fig. 10. The performance of *iBeeAIS* is comparable with that of *BeeAdHoc*. This implies that our proposed AIS extensions in *BeeAdHoc* have negligible communication and processing overhead; as a result, the performance of security enhanced protocol is comparable with original *BeeAdHoc* protocol. We can see in Fig. 10(a) that the cryptographic system, *BeeSec*, has significantly lower average throughput compared with *iBeeAIS*. The main reason of low throughput is the large overhead of carrying digital signatures in scouts and foragers to do authentication. As a result, the data transmission efficiency of *BeeSec* in Fig. 10(b) is the lowest compared with *BeeAdHoc* and its other security frameworks.

The *self/non-self* based system, *BeeAIS*, also has significantly lower average throughput compared with *iBeeAIS*, Fig. 10(a); rather *BeeAIS* has the smallest throughput among all security frameworks of *BeeAdHoc*. Our analysis shows that *BeeAIS* delivers less packets because of its inability to adapt to a changing *self* under node mobility conditions. *BeeAIS*, erroneously classifies benign scouts as malicious, because they are not seen during the initial learning phase; these scouts actually constitute the new and legitimate system *self* because of mobility. The FAR of *BeeAIS*, given in Table 8, shows that *BeeAIS* drops upto 67.36% benign scouts for a small network. If such a large number of benign scouts are dropped, the route discovery process almost comes to a complete halt. Consequently, data packets are to be dropped by the application layer because of non-availability of routes. Moreover, the transport layer considers "packet drop" as a sign of congestion and enters slow start phase. This further degrades the throughput.

Note that *iBeeAIS* and *BeeAIS-DC* that use concepts from danger theory are able to adapt to a changing *self*, and do not drop benign scouts. As a result, their average throughput is better than *BeeAIS* and very close to that of *BeeAdHoc* (see Fig. 10(a)). It is interesting to note that the *BeeSec* protocol – despite its large communication and processing cost – achieves higher throughput compared with *BeeAIS*. Our results also indicate that the network performance parameters of *iBeeAIS*, under no attack scenario, are better/comparable with those of *AODV* and *DSR* . These findings are consistent with our earlier work [38]. We can, therefore, safely conclude that the network performance parameters for our security enabled protocol are similar/better compared with state-of-the-art non-secure MANET routing protocols: *AODV* and *DSR*. This suggests that bio-inspired routing protocols enhanced with immune-inspired security framework is the next generation paradigm for developing performance efficient security solution for MANETs.

9.3. Protocol Overhead

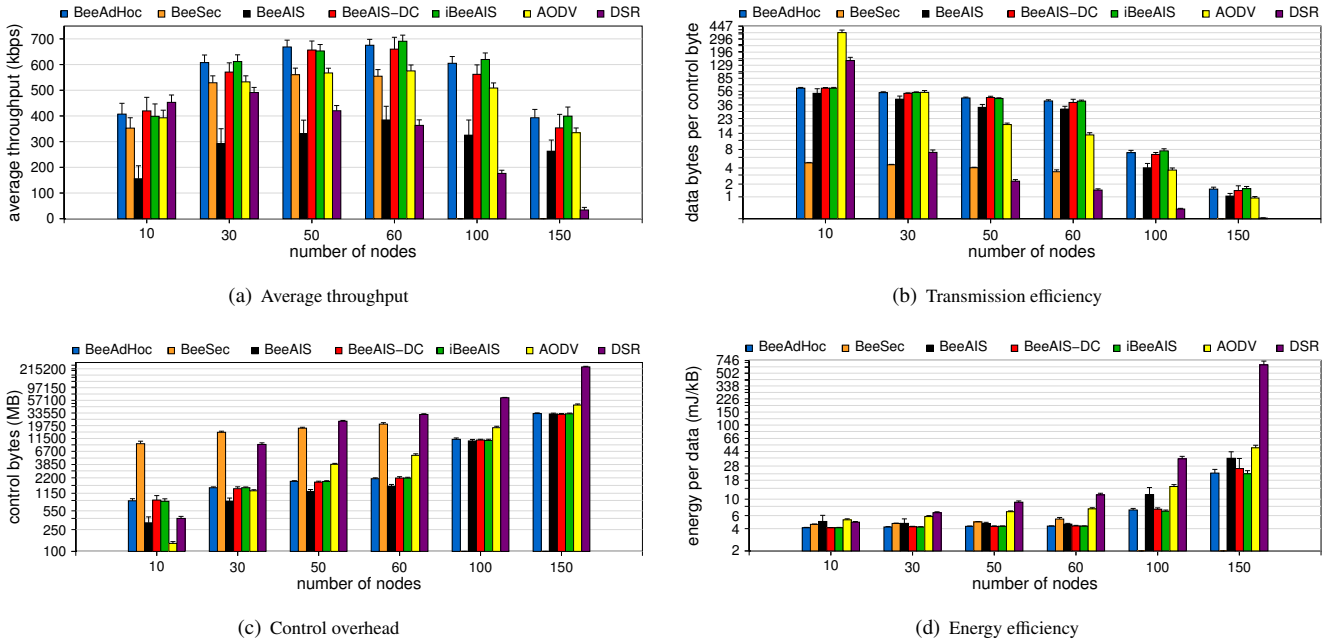
MANET nodes have constrained bandwidth and power, because they (1) *utilize wireless channels* for transmitting/receiving data and hence data rates are lower and (2) *utilize limited capacity batteries* which have limited power. Because of bandwidth and battery constraints, the costs – control and energy – of a security solution must be minimized. We have already shown that Bio-inspired security framework gives excellent performance. We now show that it is energy efficient as well. We report control overhead and energy efficiency in Fig. 10(c) and Fig. 10(d), respectively. Fig. 10(c) confirms that *iBeeAIS* transmits similar number of control bytes compared with *BeeAdHoc*. However, the control overhead of *iBeeAIS* is significantly lower compared with *BeeSec*. Moreover, *BeeSec* consumes more energy compared with the AIS based systems (see Fig. 10(d)). Therefore, we conclude that our AIS based system is more efficient in energy and control overhead as compared to the classical cryptographic security systems.

The other striking observation is that our security solution is significantly more energy efficient compared with non-secure classical routing protocols. *AODV* consumes slightly more energy compared to *iBeeAIS*, while *DSR* has significantly higher energy consumption. The control overhead of *iBeeAIS* is also lower than that for *DSR* and *AODV*, especially for large size networks.

10. Related Work

The lightweight nature of AIS based systems coupled with the dynamic coverage of antigen space [41] make AIS an ideal paradigm for protection in the domain of MANET routing. The authors in [42] [43] use *self/non-self* based AIS to secure a classical MANET routing protocol – Dynamic Source Routing (DSR) [27] – against dropping attacks only.

The danger theory and DC behavior have found useful applications in design and development of AIS based anomaly detection systems. The authors in [44] and [45] propose ways to map the immune concepts from danger theory to anomaly detection problem in computer security. They focus on identifying the different types of signals – PAMPs, danger signals, safe signals and inflammatory cytokines – and process these signals to drive the adaptive immune response. The Danger Project [46] resulted in the development of the DCA [47] and the TLR algorithm [48]. TLR closely models the BIS mechanisms of peripheral tolerance and has been applied successfully to process



(a) Average throughput

(b) Transmission efficiency

(c) Control overhead

(d) Energy efficiency

Figure 10: Comparing *iBeeAIS* performance with *BeeAdHoc*, *BeeSec*, *BeeAIS*, *BeeAIS-DC*, *AODV* & *DSR*.
(*The vertical bars in figures represent protocols in the same sequence, from left to right, as given in legend.)*

anomaly detection. DCA was introduced in [49] as an abstract model for the DCs interactions and behavior. The experimental results indicate the suitability of the algorithm for anomaly detection. The authors also conducted experiments [23] on a machine learning dataset and detection of outgoing portscans, where they concluded that DCA has the potential to act as static anomaly detector in real network environments.

Using the danger theory, the authors in [50] extended their earlier work on securing DSR to include *danger signals*. They focus on detecting malicious nodes that: (1) do not forward packets to their neighbors, or (2) do not generate a route reply from cache, with the aim to isolate such misbehaving nodes from the network. They consider packet drop to constitute a "danger" in the network and use it to improve the false positive rate. Their system does not model the DCs. In comparison, in *iBeeAIS*, we implement an accurate model of DCs to detect tampered and forged control/data packets; as a result, our focus is on countering the ways such malicious attacks change the routing pattern in MANETs. Consequently, the two systems detect different types of misbehavior, which are mutually exclusive.

The DCs have been used to secure sensor networks in [51] to detect an *Interest Cache Poisoning* attack. The algorithm – Ubiquitous DCA (UDCA) – employs a population of DCs to determine the antigen context for identifying/filtering the malicious packets. Inspired from this work, we also developed a misbehavior detection system – *BeeAIS-DC* [40] – for the *BeeAdHoc* protocol. *BeeAIS-DC* allows detection of an antigen as *non-self* only when "danger" is sensed in the network; the context is otherwise "safe" and the antigen is considered to be a *self* antigen. The system, however, could not detect forager related attacks. Our current system is a significantly extended version of *BeeAIS-DC*.

11. Extension of Integrated AIS to DSR Protocol

The principles of integrated AIS, presented in this paper, define a generic approach that can be ported to other routing protocols for anomaly detection. Such an ADS demands definition of suitable antigen structures and modeling danger signals to indicate a malicious activity. The system would then be able to perform dynamic learning of changing *self/non-self* and identify suspected *non-self* antigens by utilizing affinity maturation. In this section, we discuss the extension of integrated AIS approach to *DSR*, which is a well known classical MANET routing protocol.

DSR is a source routing protocol and hence its route discovery process resembles with that of *BeeAdHoc*. In *DSR*, routes are discovered by launching *route request* packets (the same as *forward scouts* in *BeeAdHoc*). In *DSR*, once a destination node receives a *route request*, it responds with a *route reply* message (the same as *backward scout* in *BeeAdHoc*). Route maintenance in *DSR* is different from *BeeAdHoc* in two ways: (1) It uses a specific *route error* message to indicate broken routes, (2) *DSR* maintains only a single path to a destination obviating the need of monitoring quality of discovered path. Therefore, *DSR* implements only a subset of the *BeeAdHoc* functionality. Consequently, *DSR* is vulnerable to similar routing attacks, which are launched on *BeeAdHoc*. A malicious node can use the forged/tampered route request, route reply and route error packets to subvert the normal routing behavior of *DSR*. In *DSR*, since a single path is maintained with no quality monitoring; therefore, it is possible that a single malicious packet is able to install a malicious route at the source node.

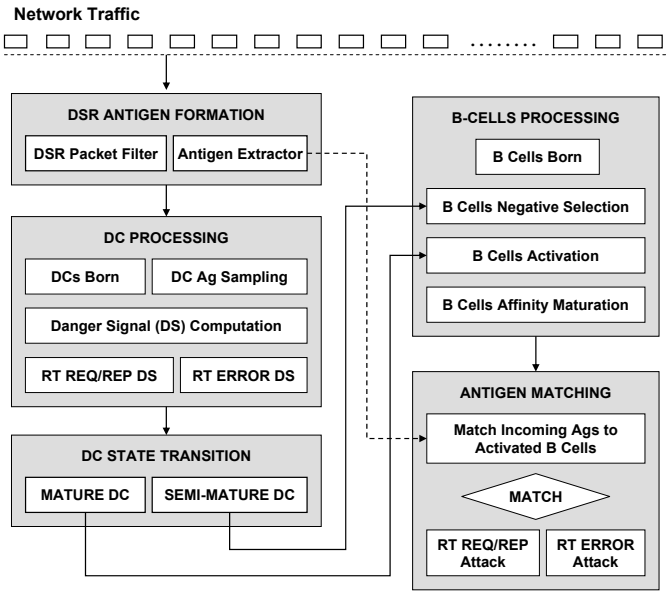


Figure 11: The Integrated AIS Framework *iAIS-DSR*

We now present the types of danger signals and antigen structures that are embedded in our integrated AIS security framework for *DSR*. The framework is named *iAIS-DSR*.

Antigens. The system uses the same antigen format as that of forager antigens in *iBeeAIS*.

$$Ag_{dsr} = \langle IP_{src}, IP_{dst}, RtLen, node_{i-1} \rangle \quad (9)$$

Danger Signals. *iAIS-DSR* requires two types of danger signals: one is related to route request/reply messages and the other is related to route error messages. The sensing of *DSR* danger signal related to route request/reply is done when a route reply is received without an associated route request. Moreover, a comparison of different routes discovered by multiple route replies received is done at the source node. The relative time/order of arrival of route replies and the length of discovered routes help in detecting the scenario when a longer path is discovered before a shorter path. Moreover, the danger signal for error messages is computed by validating the authenticity of the received *route error* message from the originating node.

System Architecture and Operation. The architecture of *iAIS-DSR* is shown in Figure 11. The system uses DCs and B-cells. The DCs sample antigens from *DSR* traffic and determine their context based on the presence or absence of danger signals. The randomly generated B-cells undergo negative selection with respect to the antigens presented in *semi-mature* context. They are activated if they match any antigen with a *mature* context. Finally, activated B-cells undergo affinity maturation to achieve a better match with the *non-self* antigens. Therefore, by adopting similar Ag/DC/B-cells processing mechanisms as of *iBeeAIS*, *iAIS-DSR* protects *DSR* against malicious attack by fabricating/tampering route request, reply and error messages.

It is important to emphasize that in *iBeeAIS*, routing attacks require sending a large number of malicious agents in the network; as a result, *iBeeAIS* can sense danger and detect *non-self* antigens while they are traversing in the network. Remember, in *DSR* routing can be subverted by launching a single malicious agent; therefore, it is a challenge to utilize effective countermeasures to restore the original routing behavior by detecting *non-self* antigens. As a consequence, *iAIS-DSR* associates danger signals with relevant antigens and then matches the antigens with the B-cells detectors database for their activation. The activated B-cells (after undergoing affinity maturation) are subsequently used to identify effected agents that are traversing on suboptimal routes. Once relevant agents are identified, *iAIS-DSR* adopts measures to restore normal routing behavior of *DSR* by replacing the optimal source route in these agents.

12. Conclusion and Future Work

The important contribution of this paper is to demonstrate the benefits of combining relevant features of the innate and adaptive immune systems. Our proposed system, *iBeeAIS*, enhances the classical AIS algorithms – *self/non-self* discrimination and DCA – and links their functionality through feedback from DCs to produce an evolving and adaptive detectors population. This enables *iBeeAIS* to perform anomaly detection in MANETs that have no stable definitions of *self* or *non-self*.

We have implemented *iBeeAIS* in ns-2 and have shown through experiments that it meets all its requirements: (1) high detection accuracy, (2) small detection delay, (3) approximately zero control overhead of the ADS framework, and (4) small processing overhead because it does not utilize complex cryptographic operations. Our comparative study with *self/non-self*, *danger theory* and a conventional ADS demonstrates the adaptive learning ability in a changing *self/non-self* MANETs environment, with significantly lower processing and communication overhead. Moreover, the network performance of secure *iBeeAIS* protocol is approximately the same (or better) as compared to non-secure *BeeAdHoc* and AODV/DSR. Our model, therefore, minimally degrades the network performance as a result of providing security solution. These properties make *iBeeAIS* a suitable candidate to secure real world MANETs. We have also also adapted *iAIS* for securing *DSR*.

iBeeAIS, can be further enhanced by utilizing more signal types – PAMPs, danger signals, safe signals, etc – to develop a sophisticated signal processing algorithm to determine the context of a DC. A system with these features can scale to a number of heterogeneous network environments. Research in this direction will be the subject of our forthcoming publications.

References

[1] M. Weiser, Some computer science issues in ubiquitous computing, Communications of the ACM 36(7) (1993) 74–84.

[2] M. Weiser, The computer for the 21st century, Scientific American 265(3) (September 1991) 94–104.

[3] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, D. Zukowski, Challenges: An application model for pervasive computing, in: Proceedings of ACM MobiCom, 2000, pp. 266–274.

[4] D. Saha, A. Mukherjee, Pervasive computing: A paradigm for the 21st century, IEEE Computer 36(3) (2003) 25–31.

[5] E. Royer, C. Toh, A review of current routing protocols for ad-hoc mobile wireless networks, IEEE Personal Communications.

[6] C. K. Toh, Ad Hoc Mobile Wireless Networks: Protocols and Systems, Prentice Hall, 2001.

[7] L. Zhou, Z. Haas, Securing ad hoc networks, IEEE Network Magazine 13 (6) (Nov/Dec 1999) 24–30.

[8] R. Bace, P. Mell, Intrusion detection systems, NIST Special Publication. SP800-31, November 2001. http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf.

[9] L. N. de Castro, J. Timmis, Artificial immune systems: a new computational intelligence approach, Springer-Verlag, London, UK, September, 2002.

[10] D. Dasgupta, An overview of artificial immune systems and their applications, in: D. Dasgupta (Ed.), Artificial Immune Systems and Their Applications, Berlin: Springer-Verlag, 1998, pp. 3–21.

[11] S. Forrest, A. Perelson, L. Allen, R. Cherukuri, Selfnonself discrimination in a computer, in: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, 1994, p. 202.

[12] S. Hofmeyr, An immunological model of distributed detection and its application to computer security, PhD Thesis, University of New Mexico, 1999.

[13] J. Kim, P. Bentley, The artificial immune model for network intrusion detection, in: Proceedings of EUFIT'99, 1999.

[14] U. Aickelin, J. Greensmith, J. Twycross, Immune system approaches to intrusion detection - a review, in: Proceedings of the ICARIS'04, LNCS 3239, Springer-Verlag, 2004, pp. 316–329.

[15] J. Kim, P. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, J. Twycross, Immune system approaches to intrusion detection - a review, Natural Computing 6(4) (2007) 413–466.

[16] P. Matzinger, Tolerance, danger, and the extended family, Annual Reviews in Immunology 12 (1994) 991–1045.

[17] P. Matzinger, An innate sense of danger, Seminars in Immunology 10 (1998) 399–415.

[18] P. Matzinger, The danger model: A renewed sense of self, Science 296 (2002) 301–304.

[19] R. Germain, An innately interesting decade of research in immunology, Nature Medicine 10 (2004) 1307–1320.

[20] R. Medzhitov, C. Janeway, Innate immunity, The New England Journal of Medicine 343 (2000) 338–344.

[21] J. Greensmith, U. Aickelin, Dendritic cells for syn scan detection, in: Proceedings of ACM GECCO'07, 2007, pp. 49–56.

[22] J. Greensmith, U. Aickelin, J. Twycross, Articulation and clarification of the dendritic cell algorithm, in: Proceedings of ICARIS'06, LNCS 4163, 2006, pp. 404–417.

[23] J. Greensmith, J. Twycross, U. Aickelin, Dendritic cells for anomaly detection, in: Proceedings of the CEC, 2006, pp. 664–671.

[24] H. Wedde, C. Timm, M. Farooq, BeeHiveAIS: A simple, efficient, scalable and secure routing framework inspired by artificial immune systems, in: Proceedings of PPSN IX, LNCS 4193, Springer-Verlag Berlin Heidelberg, 2006, pp. 623–632.

[25] N. Mazhar, M. Farooq, Vulnerability analysis and security framework (BeeSec)for nature inspired MANET routing protocols, in: Proceedings of ACM GECCO'07, July, 2007, pp. 102–109.

[26] Y.-C. Hu, A. Perrig, D. B. Johnson, Ariadne: A secure on-demand routing protocol for ad hoc networks, Wireless Networks 11 (1-2) (2005) 21–38.

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

[28] M. Zapata, N. Asokan, Securing ad-hoc routing protocols, in: Proceedings of WiSe'02, September 28, 2002.

[29] M. G. Zapata, Secure ad hoc on-demand distance vector (SAODV) routing, internet-Draft, draft-guerrero-manet-saodv-05.txt, February, 2005.

[30] C. Perkins, E. Royer, Ad-hoc on-demand distance vector routing, in: Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications, February 1999, pp. 90–100.

[31] H. Wedde, M. Farooq, T. Pannenbaecker, B. Vogel, C. Mueller, J. Meth, R. Jeruschkat, BeeAdHoc: an energy efficient routing algorithm for mobile ad hoc networks inspired by bee behavior, in: Proceedings of the ACM GECCO'05, 2005, pp. 153–160.

[32] M. Farooq, Bee-Inspired Protocol Engineering: From Nature to Networks, Springer-Verlag Berlin Heidelberg, 2009.

[33] T. Seeley, The Wisdom of the Hive, Harvard University Press, London, 1995.

[34] K. von Frisch, The Dance Language and Orientation of Bees, Harvard University Press, Cambridge, 1967.

[35] H. Wedde, M. Farooq, The wisdom of the hive applied to mobile ad-hoc networks, in: Proceedings of the IEEE Swarm Intelligence Symposium, 2005, pp. 341–348.

[36] N. Mazhar, M. Farooq, A Hybrid AIS Model for Power Aware Secure Mobile Adhoc Networks Routing Protocols, Technical Report TR-nexGINRC-2010-59, Next Generation Intelligent Networks Research Center, http://nexginrc.org/nexginrcAdmin/PublicationsFiles/TR-59-Nauman.pdf, May 2010.

[37] R. Perlman, Network layer protocols with byzantine robustness, PhD Thesis, Dept of Elec. Engg. and Computer Science, MIT, 1998.

[38] N. Mazhar, M. Farooq, BeeAIS: Artificial immune system security for nature inspired, MANET routing protocol, beeadhoc, in: Proceedings of ICARIS'07, LNCS 4628, Springer-Verlag, 2007, pp. 370–381.

[39] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: Proceedings of ACM/IEEE MobiCom, October, 1998.

[40] N. Mazhar, M. Farooq, A sense of danger: Dendritic cells inspired artificial immune system (AIS) for MANET security, in: Proceedings of ACM GECCO'08, July, 2008.

[41] J. Kim, P. Bentley, The human immune system and network intrusion detection, in: Proceedings of EUFIT'99, September, 1999.

[42] J. L. Boudec, S. Sarafijanovic, An artificial immune system approach to misbehavior detection in mobile ad-hoc networks, in: Proceedings of Bio-ADIT'04, Lausanne, Switzerland, January 2004, pp. 96–111.

[43] S. Sarafijanovic, J. L. Boudec, An artificial immune system approach with secondary response for misbehavior detection in mobile ad-hoc networks, IEEE Transactions on Neural Networks 16 (5).

[44] U. Aickelin, P. Bentley, S. Cayzer, J. Kim, J. McLeod, Danger theory: The link between AIS and IDS, in: Proceedings of ICARIS'03, LNCS 2728, Springer-Verlag, 2003, pp. 147–155.

[45] U. Aickelin, S. Cayzer, The danger theory and its applications to artificial immune systems, in: Proceedings of ICARIS'02, University of Kent at Canterbury Printing Unit, 2002, pp. 141–148.

[46] D. Project, http://www.dangertheory.com.

[47] J. Greensmith, The dendritic cell algorithm, PhD Thesis, University of Nottingham, October, 2007.

[48] J. Twycross, Integrated innate and adaptive artificial immune systems applied to process anomaly detection, PhD Thesis, University of Nottingham, January, 2007.

[49] J. Greensmith, U. Aickelin, S. Cayzer, Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection, in: Proceedings of ICARIS'05, LNCS 3627, Springer-Verlag, 2005, pp. 153–167.

[50] S. Sarafijanovic, J. L. Boudec, An artificial immune system for misbehavior detection in mobile ad-hoc networks with virtual thymus, clustering, danger signal and memory detectors, in: Proceedings of ICARIS'04, LNCS 3239, Springer-Verlag, 2004, pp. 342–356.

[51] J. Kim, P. Bentley, C. Wallenta, M. Ahmed, S. Hailes, Danger is ubiquitous: Detecting malicious activities in sensor networks using the dendritic cell algorithm, in: Proceedings of ICARIS'06, LNCS 4163, 2006, pp. 390–403.