Appl Intell
DOI 10.1007/s10489-008-0132-0

Detecting interest cache poisoning in sensor networks using an artificial immune algorithm

Christian Wallenta · Jungwon Kim · Peter J. Bentley · Stephen Hailes

Abstract The objective of this paper is to investigate how a Danger Theory based Artificial Immune System—in particular the Dendritic Cell Algorithm (DCA) can detect an attack on a sensor network. The method is validated using two separate implementations: a simulation using J-sim and an implementation for the *T-mote Sky sensor* using TinyOS. This paper also introduces a new sensor network attack called an *Interest Cache Poisoning Attack* and investigates how the DCA can be applied to detect this attack in a series of experiments.

Keywords Danger theory · Artificial immune systems · Sensor networks · Interest cache poisoning attack

1 Introduction

Danger threatens living organisms every day of their lives. Intuitively, one might therefore suppose that a successful strategy in our immune systems would be to detect danger instead of relying solely on an ability to detect the antigens that identify specific pathogens. A hotly debated hypothesis in immunology known as the Danger Theory [1] proposes just this. This theory suggests that the human immune system can detect danger in addition to antigens in order to trigger appropriate immune responses. The Danger Theory states that the appropriate immune responses produced by the immune system emerge from the balance between the concentration of danger and safe signals within the tissue of a body, not from the discrimination of self from non-self.

C. Wallenta · J. Kim · P.J. Bentley (✉) · S. Hailes
Department of Computer Science, University College London, Malet Place, London, WC1E 6BT, UK
e-mail: p.bentley@cs.ucl.ac.uk

Danger also threatens modern computer networks every day. Aickelin *et al.* [2] presented the first in-depth discussion on the application of Danger Theory to intrusion detection and the possibility of combining research from wet and computer laboratory results. Their work aimed to build a computational model of Danger Theory in order to define, explore, and find danger signals. Greensmith *et al.* [3] employed Dendritic Cells (DCs) within a Danger Theory based artificial immune system (AIS). DCs are a class of antigen presenting cells that ingest antigens or protein fragments in the tissue. DCs are also receptive to danger signals in the environment that may be associated with antigens (more details are provided in Sect. 2.2). Greensmith *et al.* abstracted several properties of DCs that would be useful for anomaly detection and proposed the DC Algorithm (DCA) to accommodate these properties. Recent work by the same authors [4] has also shown some initial results of using the DCA to detect port scanning. The outcome demonstrated the capability of the DCA as an anomaly detector.

The objective of this paper is to investigate how a Danger Theory based AIS—in particular the Dendritic Cell Algorithm (DCA) can detect one type of attack on a sensor network. The method is validated using two separate implementations: a simulation using J-sim and an implementation for the *T-mote Sky sensor* using TinyOS. For ease of reference, for the remainder of the article, the sensor network based artificial immune system will be abbreviated to SNAIS. The J-sim implementation is abbreviated as Sim-SNAIS and the hardware-specific implementation referred to as Tiny-SNAIS.

The paper continues with a presentation of the previous work related to Danger Theory based AIS. Section 3 illustrates how properties and functional requirements of sensor networks conform to an artificial tissue. Section 4 introduces

a new sensor network attack called the '*Interest cache poisoning attack*' and Sect. 5 discusses how the DCA can be applied to detect this attack in Sim-SNAIS and Tiny-SNAIS. The effectiveness of the poisoning attack and the two implementations of SNAIS are tested in a series of experiments in Sect. 6. Finally, Sect. 7 concludes the article.

2 Danger theory based AIS

2.1 Previous work

Since the first in-depth discussion of Danger Theory on the possibility of computing research [2], Bentley *et al.* [5] introduced the concept of artificial tissue in order to adapt danger and safe signals (apoptosis and necrosis) thereby triggering artificial immune responses within an AIS. The authors stressed that the tissue is an integral part of immune function, with danger signals being released when tissue cells die under stressful conditions. Related work by Greensmith *et al.* [3] employed DCs within AIS that coordinated T-cell immune responses. Kim *et al.* [6] continued Greensmith *et al.*'s work by discussing T-cell immunity and tolerance for computer worm detection. This work presented how three different processes within the function of T-cells, namely T-cell maturation, differentiation and proliferation could be embedded within the Danger Theory-based AIS. Twycross and Aickelin [7] provided a review of biological principles and properties of innate immunity, and showed how these could be incorporated into artificial models. In this work, authors addressed six properties of the innate immune system that would influence the capability of AIS. The same authors implemented the libtissue software that provides an innate immunity framework [8]. Finally, Le Boudec and Sarafijanovic [9] were also influenced by the idea of the Danger Theory, and chose to regard a packet loss in the network as a danger signal. Danger signals were used as co-stimulation signals confirming successful detection.

2.2 Dendritic cell algorithm

This paper focuses specifically on the Dendritic Cell Algorithm of Greensmith *et al.* [4, 10], which abstracted a number of properties of DCs that are possibly advantageous to design AIS for anomaly detection.

In the human immune system, during the antigen ingestion process, immature DCs experience different types of signals that indicate the context (either safe or dangerous) of an environment where the digested antigens exist. The different types of signals lead DCs to differentiate into two types: mature and semi-mature. Chemical messages known as cytokines produced by mature and semi-mature DCs are different and influence the differentiation of naive T-cells into several distinctive paths such as helper T-cells or killer T-cells. In order to employ these properties of DCs, Greensmith *et al.*. categorised DC input signals into four groups—*PAMPs* (signals known to be pathogenic), *Safe Signals* (signals known to be normal), *Danger Signals* (signals that may indicate changes in behaviour) and *Inflammatory Cytokines* (signals that amplify the effects of other signals). When each artificial DC experiences the combination of these four different signal groups released by the artificial tissue, it interprets the context of ingested antigens by using a signal processing function, which weights each type of input signal differently. The output of a signal processing function determines the differentiation status of the DC (either semi-mature or mature).

3 AIS applied to sensor networks

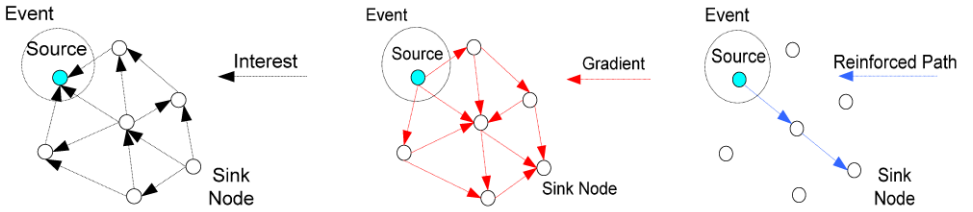
The parallels between intrusion detection and immunity have long been the source of inspiration for AIS researchers, but conventional computer networks do not closely resemble the dynamic, distributed and fluid nature of organisms and their immune systems well. There is, however, a type of network that does share many of these features: sensor networks. In the following sections, we introduce this type of network and outline one popular routing protocol, known as *Directed Diffusion* [11].

3.1 Sensor network overview

Emerging sensor networks are aimed at providing distributed and massively parallel monitoring in heterogeneous physical environments. Sensors are typically low-cost, limited capacity, mass production units, consisting of no more than (i) a sensing unit, (ii) a processing unit, (iii) memory, (iv) a transceiver and (v) a power unit [12]. Their aim is two fold: (i) to faithfully execute their intended task, and (ii) to efficiently manage their limited resources, such as energy, so as to maximise their lifetime. The following features of sensor networks distinguish them from traditional computing environments [12, 13]:

P1: Constrained resources. Limited in physical capacity, bandwidth, cost, etc.
P2: High-density. Number of nodes and density of the network can be several orders of magnitude higher than in mobile ad-hoc or wired networks.
P3: Fidelity though redundancy. Due to their physical constraints, individual nodes are prone to failure through deliberate attack or normal malfunction. The redundancy of nodes is used to compensate for this.
P4: Flexibility. Aimed at operating under diverse conditions with minimal structured support, for example deployment in remote areas.

Fig. 1 Directed diffusion [11]



(a) Interest propagation (b) Initial gradients set up (c) Data delivery along the reinforced path

P5: Dynamic network topology. The topology may change often.
P6: Frequently data centric. Networks often use data-centric routing instead of IP-like routing.
P7: Self-organising. Network connectivity is often ad-hoc and dynamically maintained.
P8: Distributed computation. Each node carries out simple data processing locally and sends out the partially processed data to other nodes.

Together, these properties have provided the catalyst for a wide range of new applications, including environmental monitoring, disaster relief operations, military control/surveillance and health monitoring [12].

3.2 Directed diffusion

In addition to the distributed and dynamic nature of sensor network hardware, one popular routing method is equally suggestive of natural immune metaphors: the *Directed Diffusion* protocol. This is a routing algorithm used to gather data sensed by a large number of sensor nodes and disseminate to a node that requests such data [11]. Directed Diffusion works in two phases, an initial exploratory phase that is followed by a reinforcement phase. Together these phases make up the three different stages discussed in Fig. 1.

The requesting node, referred to as the 'sink node' may request data from one or multiple other sensor nodes. As shown in Fig. 1(a), the sink periodically broadcasts its 'interest' packets (containing a description of the sensing task e.g. the regular reading of a patient's blood pressures) to its neighbours. Interest packets are then propagated throughout the whole network, resulting in creation of gradient fields representing the possible data flow paths from the source, back to the sink as shown in Fig. 1(b). Once the sink receives its requested data, it is then in a position to choose between its various neighbours by reinforcing the paths deemed most advantageous, for example based on the quality of service on the path that led to the neighbour, as shown in Fig. 1(c). As a result, though during the exploratory data packets are forwarded toward the sink node along multiple paths, the gradient refinement process chooses the most preferred path.

Reinforcements in Directed Diffusion come in two forms: positive and negative. Positive reinforcement encourages data flow along a given path, and the result is that data flows at a higher rate through the given path. In contrast, negative reinforcement discourages data flow along given paths, thereby reducing the rate at which data is sent through the path. The result is that the algorithm is dynamically able to tune its performance (with respect to the data flow path) based on arbitrary criteria.

3.3 Wireless sensor tissue

Readers familiar with the field of AIS should find the properties of the sensor network using Directed Diffusion very familiar, because they mirror many of the properties of AIS algorithms. In this work we regard sensor networks as a suitable metaphor for the tissue of an organism—with diffusing packets acting as signals between cells and the hardware of the sensor nodes acting as a physical embodiment of the tissue cells. Using the work of Bentley [5] and Twycross [7] to aid this analogy (and noting that not every aspect of this analogy is investigated in the paper):

- Tissue cells have limited processing, storage, and communication capacity; while a cell has its own capability of processing and storage, it takes a limited amount of input proteins such as cytokines or binds to a restricted number of neighbour cells. As described in *(P1)* sensor networks share these features.
- Biological tissue comprises a large number of cells. A tissue cell is the basic structural and functional unit, capable of functioning independently. A sensor network is similarly structured, see (*P2*).
- Each cell is prone to failure: cells in biological tissue are continuously exposed to pathogenic attacks, just as individual nodes of a sensor network are at risk, see *(P3)*. Later sections explain how an immune algorithm can integrate with a sensor network to help detect and overcome such attacks.

- The cells in living tissue move and reorganise themselves, just as nodes of a sensor network may move or be deployed in different places and have variable topologies, see *(P4)* and *(P5)*.
- Communication between biological cells is through the diffusion of signalling proteins and the matching of antigenic patterns; communication between sensor network nodes (using the Directed Diffusion protocol) is through diffusion and the matching of packets, see *(P6)*.
- Tissue cells are self-organising, growing without predetermined global control; the spatial and temporal information is passed by signals while receptors help the entire structure of the tissue develop. Likewise a sensor network automatically and dynamically forms its connectivity, see *(P7)*.
- Biological tissue cells are distributed, they work in parallel, signalling to each other to perform the desired functions. A sensor network is a truly distributed system with nodes that are processing in parallel and communicating with each other, see *(P8)*.

As discussed, the sensor network itself plays the role of artificial tissue and therefore the development of a separate artificial tissue as suggested in [5] and [7] is unnecessary.

4 Poisoning sensor networks

The analogy between sensor networks and tissue can also incorporate ideas of harm and damage. There are various types of vulnerabilities identified in sensor network environments that are often not found in conventional wired networks. This work focuses on vulnerabilities in sensor network routing protocols that rely on presence of limited capacity caches to keep a track of state of the network, for example the next hop for a packet. Directed Diffusion is one such protocol. Such protocols are typically optimised for nodes with limited resources and for specific applications, with little consideration for security.

In their seminal work Karlof and Wagner [14] analysed diverse attacks against sensor network routing protocols and introduced some countermeasures. Notable attacks discussed include: Selective forwarding, Sinkhole attacks, Sybil attacks, Wormhole attacks, HELLO flood attacks and Acknowledge spoofing. In this paper, we introduce a new attack called the '*Interest Cache Poisoning Attack*', which can easily disrupt multiple data paths in a network. The attacks discussed in [14] exploit the vulnerabilities of sensor networks that are also found from mobile ad-hoc networks. In contrast, the interest cache poisoning attack reflects the vulnerability of data-centric approaches which are often adopted for routing in sensor networks.

Under the Directed Diffusion protocol, each node maintains an interest cache that records the history of received interest packets. Each entry contains an interest and gradient(s) towards neighbouring node(s) that have sent the interest packets, such that when a data packet arrives, a node looks up its interest cache in order to find the next hop for the data. If there is a matching interest, the node forwards the data packet to the neighbour node(s) indicated by the gradient(s). Otherwise the data packet is dropped. The basic idea of the interest cache poisoning attack is to inject fabricated interest packets to replace benign entries in the interest caches of other nodes. The attack is ideally aimed at nodes on established data paths that shall be referred to as the targets of the attack.

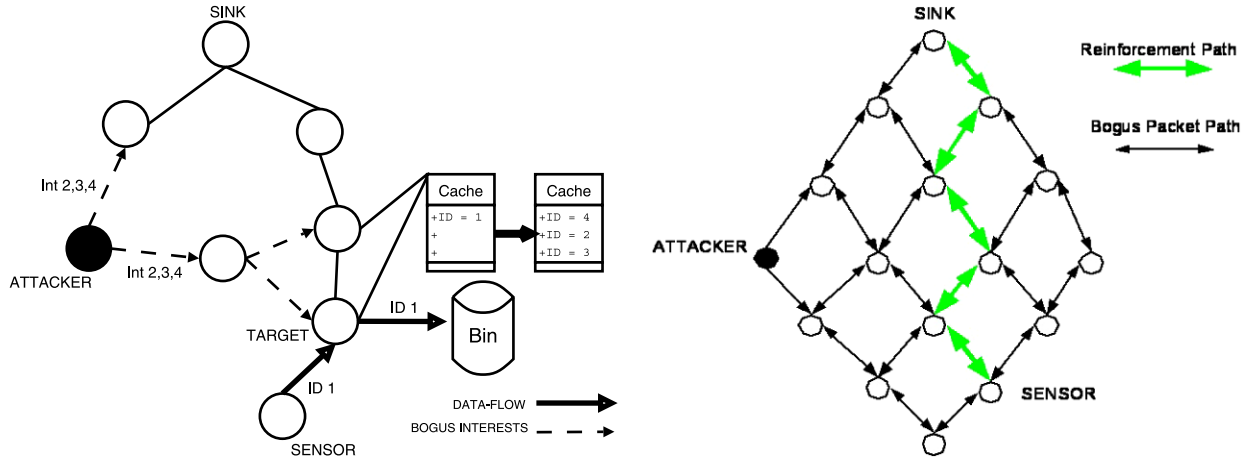
4.1 Attack mechanism and impact

A review of Tiny Diffusion (an implementation of the Directed Diffusion protocol for real sensor nodes running the TinyOS.[1]) reveals that: (i) An interest cache always has a fixed size and (ii) whenever a new interest packet arrives and the cache is full, the entry that would expire next is replaced. Therefore to realise a successful attack, the attacker can take advantage of the normal behaviour of the target by forcing it to drop the content of its cache. The attack works in two phases: First by flooding the target with bogus interests, thereby forcing it to drop those interests in its cache already. This leads to the second phase of the attack, when the requested data packets that were intended for distribution arrive, the target node is forced to drop them, since it no longer has gradients to those interested in the data.

This process will result in the disruption of data packet delivery to the sink node. Ideally, a given cache entry needs to be wiped out before the first data packet from the source node arrives at the target node. Otherwise the attack may succeed but may not be able to completely suppress the data flow. Though mechanistically different, the effect of this attack is analogous to that of '*DNS cache-poisoning*' [17]. However, we cannot use the same methods for protecting against *DNS cache-poisoning* i.e. randomised ports, restricted relaying and etc. since these are aimed at the control plane and the *Interest Cache Poisoning Attack* is performed on the data plane.

Figure 2(a) shows the impact of the attack. The attacker sends out the bogus packets and fills up the cache of the nodes on the data path. The bogus interests will replace the original interest with ID 1. When the requested data with ID 1 arrives later, the target node will just drop it. This is because there is no matching entry in the cache. As shown in the Fig. 2(b), the attack will even be successful if the attacker is not next to the target node. The attack exploits the flooding behaviour of Directed Diffusion. Whenever a node

[1] TinyOS is an open-source operating system designed for wireless embedded sensor networks (http://www.tinyos.net/).



(a) Interest Cache Poisoning Attack Overview (b) Bogus interest packet propagation

Fig. 2 The interest cache poisoning attack

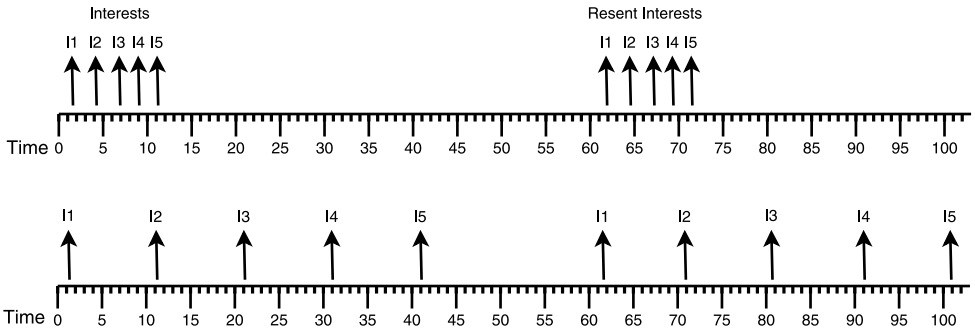


Fig. 3 Subscriptions sent out at once and with a gap between two consecutive interests

receives a new interest packet it will rebroadcast it to all its neighbours. Hence, the bogus interest packets are spread and affect the caches of many nodes, eventually the cache of a target node. As a result, the impact of bogus packets can propagate over an entire network and disrupt multiple paths of data packet delivery.

4.2 Normal behaviour of directed diffusion

SNAIS can be classified as an anomaly-based IDS. For an anomaly detection algorithm, it is important to define the normal behaviour since the anomaly detector regards deviations from normal behaviour as an attack. The problem we encounter with Directed Diffusion in defining normal behaviour is that there is no standard application. There are many scenarios and applications with different requirements and behaviours where Directed Diffusion operates. This makes the generation of 'normal' data sets difficult. In order to tackle this difficulty, this work defines some normal behaviour for Directed Diffusion. This normal operation is mainly determined by the behaviour of the sink node. This is because the behaviour of a sensor node is well-defined and is determined based on the behaviour of the sink node. The sink node can change several parameters that affect the normal operation, and thus the signals generated by the intrusion detection system (see Sect. 5.2.2).

- The **interest expiration** determines how long an interest is stored in the cache. If an entry is overwritten before it expires, the IDS will conceive this as deviation from normal behaviour (see danger signal DS2).
- The **number of subscriptions** affects the load of the system. More subscriptions lead the interest cache to have more interest cache entries on average. Hence, an attack causes more damage and creates more danger and PAMP signals.
- **Time between multiple interests**: If the sink node sends out all interests at once, they expire and need to be re-sent at the same time. Alternatively, the sink node can send out multiple entries over a certain interval with a gap between two consecutive packets. Figure 3 shows both options. We use the cache update rate to generate danger and safe signals (see danger signal 1 and safe signal 3).
- The sink can change the **order of the interests**. This is interesting if the sink is aware that some malicious behaviour is going on. The first interest that arrives at a node is most likely the one that is overwritten first. Therefore, subscriptions with a higher priority could be sent later

than interests with a lower priority. On the other hand, interest packets experience different delays when travelling different paths, which may change the order anyway.

The sink can either **change the subscriptions** frequently or maintain the same subscriptions over a long period. If the sink node changes the subscriptions often, the data paths are likely to change as well and hence the nodes on the data paths change too. This affects the generation of safe signal and inflammatory cytokine on these nodes and the nodes on the previous paths. Furthermore, the attack impact changes if the attacker's distance to the new data paths has changed too.

4.3 Design of the interest cache poisoning operation

Section 4.1 introduced the general idea of the interest cache poisoning attack. This section describes the various types of the attack. The aim is to identify significant attack parameters that will be evaluated in Sect. 6.

4.3.1 Attack parameters

The attacker can collect various information in order to perform a successful attack, for example, to achieve a high data suppression rate and a low probability of being detected by the intrusion detection system. The following list is important information for an attacker in order to set the right attack parameters.

- An attacker that is aware of the intrusion detection system would set the **interest expiration time** to a short value. Then the DCA has less time to classify the packet because it is removed from the cache quickly. However, if the expiration time of the bogus packets is too short, the attack might fail. This happens if the cache update rule is designed to replace the entry with the shortest remaining expiration time. If the bogus packets have a shorter expiration time than the benign packets, the attacker will overwrite its own packets and the attack will fail. Thus, the attacker needs to be aware of the interest expiration time set by the sink node in order to set the expiration time of the bogus interest packets correctly.
- The **TTL (Time-to-Live)** field of an interest packet determines the number of hops that the packet travels before it is dropped. If the attacker is aware of the default TTL that the sink node sets to its interests, he can estimate the distance between himself and the sink node. This is useful information if the attacker does not want the bogus packet to approach the sink node, because he expects the sink node to become suspicious about the bogus interest packets. The attacker can set the TTL of his bogus interests to a small enough value and the bogus packets will be dropped before they arrive at the sink node. As the TTL affects the number of hops that a bogus packet travels, it also affects the impact of the interest cache poisoning attack. Bogus packets with smaller TTL arrive at fewer nodes, and hence fewer interest caches are poisoned.
- The **number of bogus packets** influences the success of the attack and the energy used by the attacker. If the attacker is not interested in suppressing the entire data packets of all sink subscriptions, fewer than 'cache size' packets can be sufficient. For example, if the benign interests of the sink node completely fills the interest caches of the nodes, a few bogus interest packets are enough to wipe out some of the benign entries. Moreover, if the attacker is aware of the intrusion detection system, fewer bogus packets increase his chances of not being detected by the IDS.
- The **position of the attacker** is not a parameter that the attacker can influence directly. However, different attacker positions will affect the success of the attacks. An attacker close to the established data paths or close to the sensor nodes, which generate the requested data, will most likely suppress more data packets. If all the subscribed sensors are located close to the sink and the attacker is many hops away from them, it takes relatively long to complete the attack. Hence, more data packets will be delivered to the sink node and the attack will not be so successful.
- The attacker can change the **sender node ID** when sending out packets. This will prevent the linkage of malicious behaviour to a certain node ID.
- The **interest attributes** determine whether a bogus interest packet results in data generation or not. An attacker that is aware of the IDS might want to generate bogus interests that result in data being delivered to him, because interests that generate data cause the release of safe signal and appear less suspicious. On the other hand, receiving many data messages consumes the attacker's energy. However, the attacker needs to be aware of the addressing scheme of the network in order to send correct interests. He cannot re-send the benign packets, sent by the sink node, because those will either be dropped as duplicates or just set up a second gradient towards the attacker without wiping out the benign entries. Also, slightly changing the attributes of the benign packets is not necessarily successful if the addressing scheme is not known. For example, if X is an invalid attribute in the addressing scheme, that does not mean that $X + 1$ is valid as well.
- **Attacking specific data streams**: As the sink might have prioritised the requested data, the attacker might be interested in suppressing specific streams with a high priority. Hence, if he does not want to wipe out all the packets in the cache, fewer bogus packets might be sufficient.
- **Time between multiple bogus packets**: The attacker can either burst out the bogus interests very quickly or leave

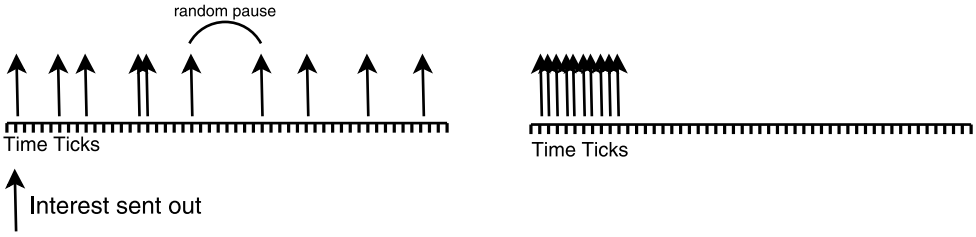


Fig. 4 Normal sending behaviour (*left*) and bursting packets (*right*)

a pause between two consecutive packets. For example, the Tiny Diffusion implementation [15] leaves a pause between two consecutive packets in the sending queue. This pause is randomly chosen between $t_{min} = 125$ ms and $t_{max} = 1000$ ms. If the attacker sends out his bogus packets faster, they arrive at the target nodes earlier, and hence more data packets are dropped because the attack is completed earlier. On the other hand, the attack might be detected by the IDS because this is a deviation from the normal sending out behaviour. Figure 4 shows both sending behaviours.

- **Attack time**: The attacker needs to figure out when to start sending the bogus packets. An early attack might be less successful, because the bogus packets might be overwritten by benign packets again and a late attack might be less successful, because too many data packets are already delivered to the sink node. Finding the right attack time is discussed in more detail in the following sections.

Many variants of the interest cache poisoning attack are possible by changing the attack parameters described. However, this work cannot investigate all these possible attacks. Our attacker uses the same interest expiration time as the sink in order to avoid overwriting his own entries. He also uses the default TTL value and does not care whether the bogus packets arrive at the sink node or not. The attacker sends out as many bogus packets as the interest cache has cache lines. Fewer packets would cause less damage and more packets increase the risk of being dropped due to full sending queues. As already mentioned, the position of the attacker is chosen randomly. The attributes of the interest packets are filled with random data. The attacker does not attack specific streams, because he does not know the sink node's priorities on data streams. The attack time and the time between multiple bogus packets are discussed in more detail in the next section when three attack variations are described.

4.3.2 The simple burst attack

The burst attack is a simple form of the interest cache poisoning attack. Every time the attacker receives a new interest, he sends out at least as many packets as a cache has lines, maybe even more. The advantage of the attack is that it is very simple and does not require the collection of any information before it can be performed. The drawback is that it uses a lot of resources, because the attacker has to send out a lot more packets than actually required. If the sink sends out all its interests at once, the attacker can wait until the last interest has been sent out before sending the bogus packets. This would reduce the packet overhead and save the attacker energy. Another issue is that the sending queues of the nodes are limited and such an amount of packets might cause many nodes, including the attacker, to drop packets because the sending queue is full. The behaviour of the simple burst attack might easily be detected by a simple burst detector. A detector could compare the sending behaviour of the attacker with the sending behaviour of a normal node and raise an alarm if an anomaly is detected.

4.3.3 Improving the attack

The first step to improve the attack is to reduce the packet overhead. To achieve this, the attacker has to determine when to send the bogus packets. With the assumption that the sink node sends all its interests at once, the attacker needs to determine when the last interest has been sent. He could start a timer whenever an interest packet is received and start his attack when there has not been any packet for a time t. Another way is to measure the cache updates for the last n seconds using a sliding window technique similar to the one used to calculate the danger signal DS1 (see Sect. 5).

The attack that uses this technique but still sends out the bogus packets without the random pause between two packets is referred to as the fast attack. If the attacker shows the same sending behaviour as any other node, in other words inserts the random pause between two consecutive packets, the attack is slowed down but the detection probability is likely to be decreased. This attack is called the slow attack. Yet another attack sends out bogus packets constantly with a gap between two consecutive packets. This is referred to as the constant attack. All these attacks have a trade off between being detected and the damage caused by the attacks. Section 6 will examine more closely the effectiveness and efficiency of these four variations of the interest cache poisoning attack. Clearly, there are many more variations of the interest cache poisoning attack.

4.3.4 Attacking the IDS mechanism

This work does not analyse attacks on the intrusion detection system itself. These kinds of attacks are possible and can have an impact on the system, especially the automatic response. If the attacker is aware of the IDS and knows how it works, he can try to make benign packets look dangerous so that the packet filter will drop them. This form of attack is beyond the scope of this work.

5 SNAIS design

The following sections detail the design of the sensor network based artificial immune system (SNAIS), giving all design assumptions, algorithm, signals, and implementation details. Sim-SNAIS and Tiny-SNAIS are both Intrusion Detection Systems that share the same algorithm (the Dendritic Cell Algorithm), but the different hardware and software constraints imposed by J-sim and TinyOS mean there are minor differences in signals and implementation. Further details are available in [3, 4, 15].

5.1 Assumptions

There are two main reasons for making assumptions. Firstly, no system can provide a perfect solution to every possible situation or application. Indeed, this work does not claim to detect all existing attacks. Thus, we make assumptions that limit the network environment. The IDS in this work aims to provide a solution to such a limited environment. Secondly, simulation environments have several constraints and this makes assumptions necessary. These assumptions are described in the following sections.

5.1.1 Link layer assumptions

Links between nodes are bidirectional and symmetric, although links in wireless sensor networks are usually asymmetric and lossy. If two nodes maintain a link, the bit error rate for this link is 0% and so the link is not lossy. However, packets might get dropped, for example, because of packet collision. The intrusion detection system does not require the nodes to operate in promiscuous mode, but it might be necessary for future extensions. A node in the promiscuous mode is able to overhear packets sent by neighbour nodes. One example where the promiscuous mode is needed is the watchdog mechanism [16]. The watchdog overhears packets and hence can monitor the forwarding behaviour of neighbour nodes.

5.1.2 Sensor node assumptions

The nodes are low-cost sensor nodes with small capacity CPU and memory and limited battery power. Each node has a unique ID, but the node ID assignment is not secured by any central authority. Hence, changing one's own ID and impersonating other sensor nodes is possible. Impersonating the sink node is also possible because even messages from the sink node are not authenticated. Compromising the sink node itself is not possible, because it maintains stronger security mechanisms than the sensor nodes. Nodes do not use the sleep mode: they are always able to receive and send packets. A node trusts its own hardware (e.g. clock or sending unit) and is aware of whether it is able to respond to a sensing task or not. Nodes do not share any key material with other nodes or the sink node. Packets are not authenticated or encrypted. Therefore, nodes cannot trust each other, and the packets they receive cannot be verified. Each node has only a local view of the network and is only aware of its neighbours.

5.1.3 Network traffic assumptions

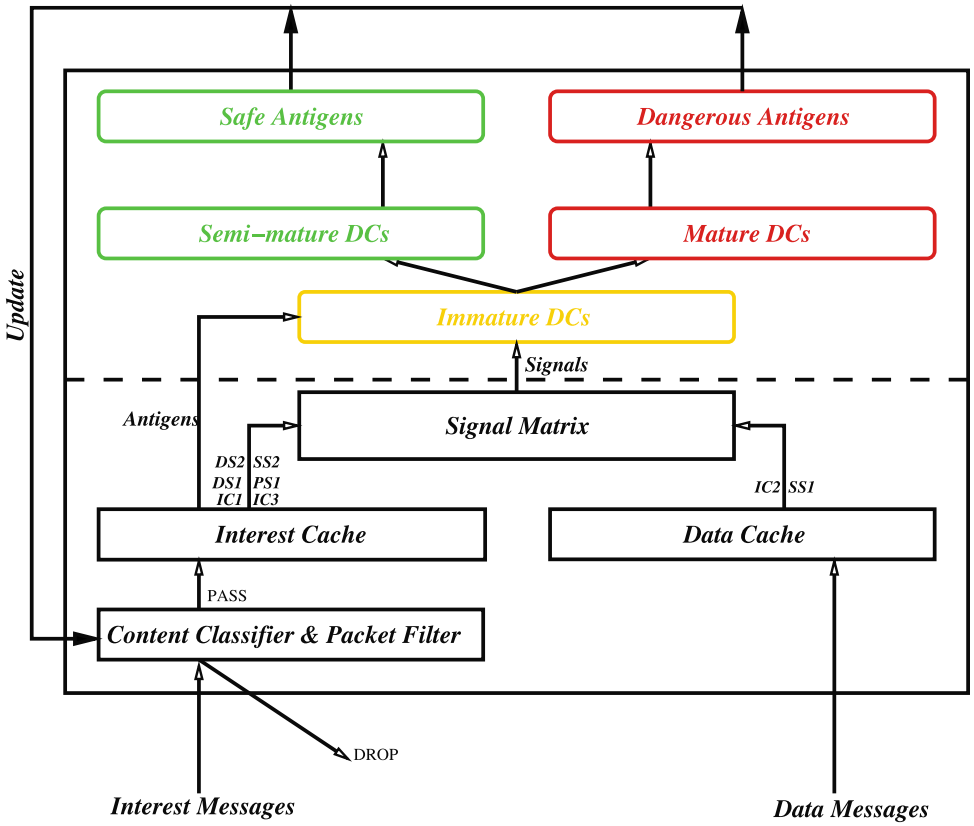
In Sim-SNAIS, Directed Diffusion has been implemented using J-sim. In Tiny-SNAIS, Tiny Diffusion, a simplified version of Directed Diffusion implemented by researchers at UCLA,[2] is used, providing an adequate framework to evaluate the intrusion detection system design. There are only two message types: (i) interest messages sent by the sink node and (ii) data messages generated by sensor nodes. Details of the Tiny Diffusion implementation are given in [15]. All traffic is treated equally so there is no critical or prioritised traffic. The communication primitives are broadcast for interests and unicast for data messages. The sink broadcasts interests periodically and then the sensors generate and forward the data packets back to the sink node.

5.1.4 Network assumptions

The number of nodes in a network lies between 10 and 100. The sensor nodes are randomly deployed in a fixed-sized area. The topology does not change during the operation time of the network. No nodes join or leave the network and no nodes fail. The number of neighbours for each node is in the order of 5–10 nodes. There is no hierarchy between the nodes except the distinction between sink and sensor node. Nodes do not build up clusters or filter, aggregate or summarise data or interest messages. There is only one sink node that does not change its position. The intrusion detection system will operate on all nodes apart from the sink node and the compromised node.

[2] The Tiny Diffusion project. Center for Embedded Network Sensing, http://www.cens.ucla.edu/ eoster/tinydiff/.

Fig. 5 System overview



5.1.5 Attacker assumptions

The design of the intrusion detection system focuses on the newly introduced interest cache poisoning attack, which is a denial of service attack on the routing layer. The attacker may physically attack and compromise a node or send malicious code to change the behaviour of a node. Once he has compromised a node, he is able to control that node. For instance, the attacker can extract all the information stored on the compromised node or can re-program it. The attacker can change the sender node ID of a packet and impersonate other nodes, even the sink node. The compromised node still has the same resources as a normal node and has no possibility of collecting global information. There is only one compromised node and this node is chosen randomly from within the network. The attacker does not change the position of the compromised node. Regarding the interest cache poisoning attack, the bogus packets sent out by the attacker are slightly different from those that the sink node sends. The attacker does not care whether a bogus interest will result in actual data delivered to any node. He is aware of the structure of an interest message, but the attributes that describe the sensing tasks are filled with random data.

5.2 SNAIS design

5.2.1 System overview

Dendritic Cells act as intrusion symptom collectors and detectors, and T-cells are responsible for an automated response. Inspired by this behaviour, the intrusion detection system has two aims: (i) detect an interest cache poisoning attack and (ii) trigger an appropriate response to an attack. Figure 5 illustrates the system overview. The main components of the system are explained in more detail in the following sections. According to the biological model, the Dendritic Cells act as detectors. An antigen is represented by interest packets that are stored in the interest cache. Immature DCs capture antigens from the interest cache. The data cache and the interest cache are the sources for the various signals collected by the DCs. These signals indicate normal and abnormal situations and the generated signals are then stored in the signal matrix. When a new antigen arrives, a new immature DC is created and captures the antigen. Immature DCs then copy the signals from the signal matrix into their own signal store. When an immature DC has received enough signals, it differentiates then into either a semi-mature or a mature DC. The captured antigen is classified as either dangerous if the DC is mature or safe if the DC is semi-mature.

Responding to the attack is done by filtering out interest packets before they enter the system. The problem is that the IDS cannot know in advance if a packet is dangerous or safe. The basic idea is to extract generalised patterns from packets that have been classified by the Dendritic Cells, and to use this information to filter out new packets depending on their patterns. With the information from the classified antigens, the filter is updated. Therefore, a content classifier, which classifies the content of a new packet, and a packet filter that drops packets depending on their content, is needed. Overall, the system is a stand-alone intrusion detection system because the nodes do not exchange any information about intrusions. This design decision is made because nodes do not share any secret keys, and thus signing and verifying such information is not possible. Without the possibility of authentication, a cooperative IDS is always vulnerable to false claims and accusation.

5.2.2 Signals

The Dendritic Cell Algorithm within SNAIS uses four different types of input signals: PAMP signals, danger signals, safe signals and inflammatory cytokines. The following sections describe suggested signals that are collected from a Directed Diffusion based sensor network. Safe signals are designed to indicate normal operating situations, while danger and PAMP signals are designed to indicate the presence of an interest cache poisoning attack.

PAMP signal (PS)—*Generated from data delivery failures* A PAMP signal (PS) is a strong indicator of a pathogenic presence. Hence, the signal collected from the network environment should strongly indicate the presence of an attacker. The failure of data delivery to the sink node can be such an indicator. Delivery failures may also result from node failures on the established path or the absence of sensor nodes generating the requested data. The PAMP signal, however, definitively establishes that what was expected did not happen and can be used to launch further investigation. This relative difference of confidence in abnormal behaviour makes the PAMP signal stronger than a danger signal. For this purpose, the failure of requested data delivery would cause the sink node to generate a PAMP signal. Unlike other signals that are just generated locally and not forwarded to other nodes, the PS is forwarded to other nodes. In order to transport the PAMP signal, a re-sent interest packet is used. When the sink node subscribes to a data stream, it expects as many packets in one refresh period as specified in the data rate of the interest packet. For example, if the data interval is set to 10 s and the refresh interval is set to 60 s, the sink node expects to receive 6 data packets within one refresh interval. In Tiny-SNAIS, the concentration of the PS is given as the fraction between the number of expected data packets $n_{expected}$ and the number of received data packets $n_{received}$:

$$PS_{conc} = 1 - \frac{n_{received}}{n_{expected}}.$$

Any node receiving the refreshed interest with $PS_{conc} > 0$ inserts the packet into the interest cache and the actual PS is generated later when this interest cache entry is overwritten.

In Sim-SNAIS the basic definition of PS is the same, however, the calculation of $n_{expected}$ and $n_{received}$ is slightly different from that used in Tiny-SNAIS. Sim-SNAIS uses the full version of Directed Diffusion, which sets up the reinforcement paths for data delivery. In Tiny-SNAIS, $n_{expected}$ is calculated by dividing the interest refreshing rate by the data rate. The one phase pull version of Directed Diffusion, used by Tiny-SNAIS, has only one data rate since there is no separate exploration stage and data delivery stage. The full version of Directed Diffusion, which is used by Sim-SNAIS has two different data rates: a higher rate for delivering the data packet at the exploration stage and a smaller rate for delivering actual data at the data delivery stage. In Sim-SNAIS, $n_{received}$ and $n_{expected}$ is counted as follows.

Whenever a sink node receives the exploratory data packet (data packet matching to the sent interest with a bigger rate equal to 50), it sends out a positive reinforcement packet, meaning that sets up the positive reinforcement data delivery path. At this moment, $n_{expected}$ is calculated as:

(*the next interest refresh time*
  − *the time that the positive reinforcement packet is sent out*)
/(*the data rate*)

In addition, at this moment, IDS module at the sink node sets up the data counting timer and starts counting the data packet delivered to the sink node. This counting continues until the refreshed exploration interest is sent out again by a sink node. The counted value becomes $n_{received}$. When the count of $n_{received}$ is finished, PS_{conc} is calculated and added to the refreshed interest as an additional attribute and sent out. One final consideration was needed to generate the PS in the full version of Directed Diffusion. As soon as the positively reinforced path is negatively reinforced, $n_{received}$ is counted no longer and PS_{conc} is not calculated either. Hence, PS is not generated.

Safe signal 1 (SS1)—*Generated from data packet arrival* This signal shows that the data requested by the sink node has been forwarded to a given node. The nature of the safe signal is to indicate normal operation of Directed Diffusion. The absence of a safe signal does not necessarily indicate

the existence of an attack, but a safe signal aims to suppress a false detection alert. The entry of a data cache, which records the data packet forwarded, would serve this purpose. Whenever a data packet that matches an interest in the interest cache arrives, it will be forwarded and recorded in the data cache. Therefore, whenever a new entry is inserted into the data cache, a safe signal is generated and the concentration is simply 1. It has to be taken into account that this safe signal is only generated if the node is on the reinforced data path of this particular data stream.

Safe signal 2 (SS2)—*Generated from cache entry expiration* The second safe signal, the normal expiration of an interest cache entry, is the counterpart of the danger signal DS2. It indicates the normal, regulated 'death' of a cell, represented in Tiny-SNAIS by an interest cache entry. Whenever a cache entry expires normally or is just refreshed by the same interest, the safe signal is released. The concentration of the SS2 is simply 1.

Unfortunately, this process also works for bogus packets, i.e., bogus packets remaining in the interest cache also expire. Thus, too much of this signal is generated when an attack is on-going. Therefore, in Sim-SNAIS the definition of this signal is slightly modified. In Sim-SNAIS, SS2 is generated when the interest cache entry is refreshed (not overwritten). In general, an attacker does not want to send exactly the same bogus packets over and over because it can increase the chance of being detected. Thus, SS2 generation whenever the interest packet is refreshed appears to be a reasonable modification. The concentration of SS2 is simply 1.

Danger signal 1 (DS1)/safe signal 3 (SS3)—*Generated from cache update rate* The interest cache update rate is used to calculate either the first danger signal DS1 or the safe signal SS3. A normal cache update rate will generate a safe signal while an abnormally high update rate will release a danger signal. Given C, the cache size, any number of updates n that is less than or equal to C for the last T seconds is normal. Any number higher than that indicates danger since the interest can be overwritten immediately. For the sink node, it does not make any sense to send more updates than a cache can handle within a short time. Hence, if the number of updates is greater than C, the danger signal DS1 is released. The concentration of the two signals is calculated as follows:

$$SS3_{conc} = 1 - \frac{n-1}{C};\ 0 \le n \le C,$$

$$DS1_{conc} = \left(\frac{n}{C}\right)^2;\quad n \ge C.$$

With a higher number of updates, the safe signal will decrease linearly until it reaches the threshold C, while the DS1 increases with the power of 2 to emphasise the abnormal behaviour. To count the number of cache updates, a sliding window is used. The window has two parameters: (i) the window size T in seconds and (ii) the sliding size r. The intrusion detection system will constantly count the number of updates for the last T seconds and the window is shifted every r seconds. For example, a window with $T = 10$ and $r = 1$ counts the packets within the last 10 seconds and shifts the window every second by 1.

Danger signal 2 (DS2)—*Generated from cache entry overwriting* There are two ways for an entry to be removed from the interest cache: (i) when the entry expires, or (ii) when the cache is already full and it is replaced by a new entry. Although a sink is able to overwrite its own entries in a cache by carelessly sending a large number of different interests during a short interval, we do not expect this behaviour to be the norm. Therefore, the overwriting of entries long before their expiration time can indicate the presence of an attack. In order to identify such an event, the expiration field is checked when an entry is inserted and when it is overwritten. The concentration of a DS2 signal is the difference between the remaining time $T_{remaining}$ of an interest and its original expiration time $T_{expiration}$. Overwriting a very recent entry will lead to a much stronger signal than overwriting an early expired entry.

$$DS2_{conc} = \frac{T_{remaining}}{T_{expiration}}.$$

5.2.3 Signal matrix

Signals need to be stored in a data-structure where they are easily accessible for the Dendritic Cells. Greensmith *et al.* [4] suggest the use of a signal matrix to represent the signals that are released into the tissue. Their matrix contains entries for all the different signals. Whenever a signal is generated, it is stored in the signal matrix. In every cell cycle, a Dendritic Cell copies the signals from the matrix into its own signal store. The copying models the signal collection by a DC from the environment.

The first design choice regarding the signal matrix is the general structure of the matrix. Greensmith *el al.* use a simple signal matrix with one entry per signal. This simple matrix is shown in the top figure of Fig. 6. Another option is a signal matrix that has as many rows as interest cache entries. This is shown in the bottom of Fig. 6. The larger matrix makes it possible to link some signals directly to cache entries. For example, if a new interest packet overwrites an old entry, the danger signal DS2 is released. Using the larger signal matrix, this danger signal is only stored in the matrix row linked to this cache entry. Moreover, only the Dendritic Cell that captured this interest entry copies this danger signal into its own signal store.

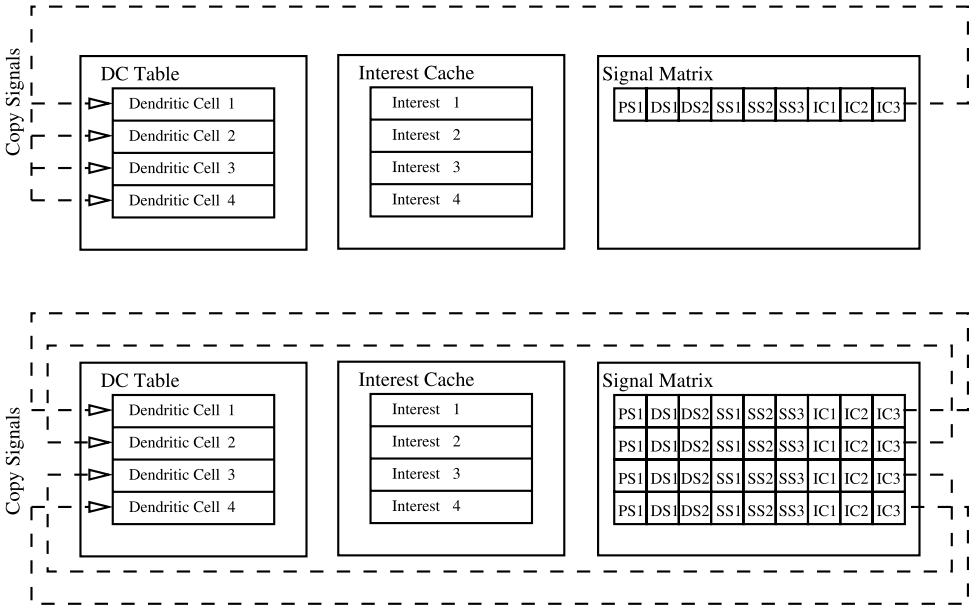


Fig. 6 Two options for the signal matrix structure

(In Sim-SNAIS the signal matrix with multiple rows is implemented in a slightly different way. A signal lifespan is used, which limits how long each signal can remain in the matrix. For multiple rows, all signals stay as long as their ages do not reach their lifespan. So, in theory, there is no limit in the number of rows for the signal matrix. This implementation of the big matrix was performed in order to investigate any side effects caused by overwritten signals.)

The update rule of the signal matrix determines how the signal matrix is updated when new signals are generated. The original Dendritic Cell Algorithm simply overwrites the old signal when the same signal type is generated again. Another option is to overwrite an old signal, only if the new signal has a stronger concentration to avoid situations where a signal with a strong concentration is quickly overwritten by one with a weaker concentration and not collected by any DC at all. Yet another possibility is to store the last n signals of each type. This option requires more memory and is therefore not used.

Signals can either be stored in the matrix until they are overwritten, or they can be aged and expire after a certain time. This prevents a signal from being in the cache for a very long time if no new signal of this type overwrites it. Another option is to weaken the concentration of a signal each time interval. This system uses a maximum signal age for each signal. After a signal has expired, it is removed from the signal matrix. This mechanism is simpler than weakening the signals but still prevents a signal from being stored in the cache long after it was caused.

5.2.4 Antigens

In vivo, antigens are collected by Dendritic Cells and classified as dangerous or safe depending on the balance of the collected signals. In a Directed Diffusion based system, interest packets, data packets, interest cache entries and data cache entries could be used as antigens. Regarding the interest cache poisoning attack, the data packets and data cache entries are not a threat for the system. Furthermore, interest messages that are dropped as duplicates before they enter the cache do not harm the system. Hence, interest cache entries are used as antigens.

The cache is filled with benign antigens represented by the interest messages sent by the sink node, and malicious antigens (pathogens) represented by the interest messages sent by the attacker node. The task of the DCs is to capture those antigens and classify them correctly as safe or dangerous.

5.2.5 Dendritic cells

A natural Dendritic Cell captures antigens and collects signals which have been released into the tissue. After being exposed to a sufficient number of signals, an immature DC will differentiate into either a semi-mature or mature DC. Captured antigens are presented to immune cells that trigger an appropriate response. Antigens presented by a semi-mature DC are regarded as safe, whereas antigens presented by a mature DC are regarded as dangerous.

Figure 7 shows the work of the artificial DCs in the intrusion detection system. An immature DC is idle until a new interest message is stored in the cache entry that the DC is linked to. This event causes the immature DC to start its work. It captures the antigen represented by the new cache entry and starts collecting signals from the environment. This is modelled by periodically copying signals from the signal matrix into the input signal store of the DC. Every

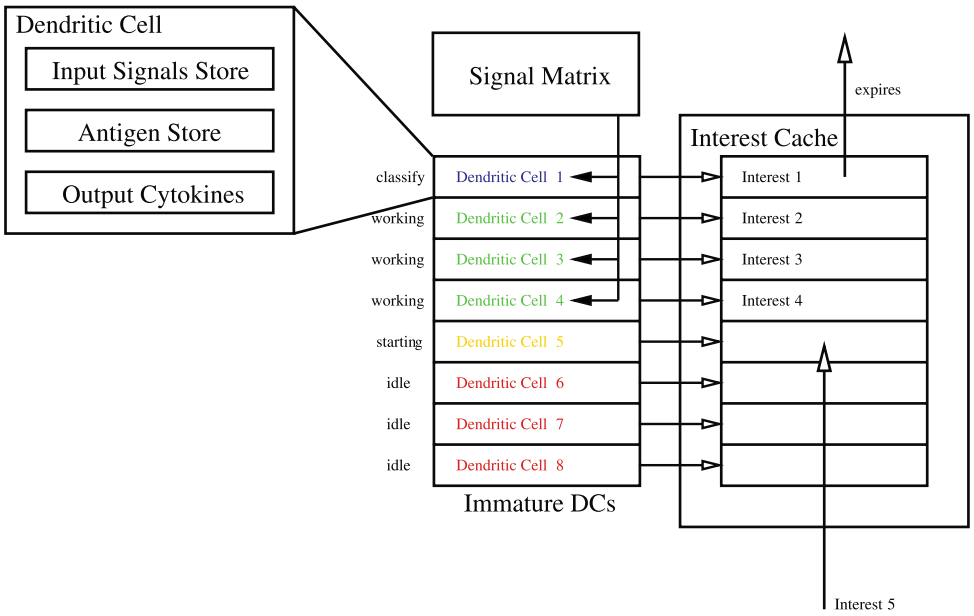


Fig. 7 Dendritic Cells working on the interest cache

time signals are copied, the DC calculates the temporary output cytokines using the formula described in [15]. This temporary output cytokine is accumulated and stored in the DC.

When the interest cache entry is removed from the cache, the DC that captured this cache entry makes its decision about this entry. The output cytokines (o_x) show whether the DC is mature ($o_{mature} > o_{semi\text{-}mature}$) or semi-mature ($o_{semi\text{-}mature} >= o_{mature}$). A mature DC indicates that the captured interest message is dangerous while a semi-mature DC indicates that the interest message is safe. There are several design choices regarding the design of the Dendritic Cells:

1. The total number of DCs that are deployed by the intrusion detection system.
2. The antigen capacity determines how many antigens a single DC can capture.
3. If a single antigen is captured by multiple DCs (multiple sampling) or not.
4. The maturation time of a DC, in other words, when does the DC become mature or semi-mature?
5. The cycle rate of a DC determines how often the signals are copied from the signal matrix into the signal store of the DC and how often the temporary output cytokines are calculated.
6. The signal weights that determine the contribution of each signal to the output cytokines.

When Greensmith *et al.* (2006) applied their algorithm to detect 12 port scanning, they used a DC population of 100. Each DC sampled only one antigen, but a single antigen could be sampled by multiple DCs. A DC differentiated when the costimulatory cytokine, a cytokine that indicates how much signals have been received so far, exceeded a certain threshold that was randomly chosen from a given interval. This randomisation caused the DCs to maturate at different times and collect different numbers of signals. A final decision about an antigen was made by a majority vote of all DCs that captured this antigen.

The use of the Dendritic Cell Algorithm on sensor nodes forces the algorithm to be as simple as possible while still being effective. To create a lightweight system, in SNAIS each DC captures only one interest cache entry and an entry is not sampled by multiple DCs. Thus, the total number of DCs will be as many as the total number of interest cache entries. The signal collection of a DC is done by simply copying the signals currently stored in the signal matrix into the internal signal store of the DC.

Natural Dendritic Cells differentiate after they have been exposed to a sufficient number of signals, but it is not yet completely understood how much signal is needed to launch this process. The original Dendritic Cell Algorithm uses the costimulatory cytokine to decide the maturation time of a DC. This cytokine indicates the number of signals a DC has already collected. When the cytokine reaches the threshold, the DC becomes either semi-mature or mature. This cannot be applied to the intrusion detection system for sensor networks because an interest message might be removed from the interest cache before the costimulatory cytokine reaches the threshold. For example, cache entries can be overwritten very quickly by another entry. Therefore, in SNAIS a DC maturates when the captured interest cache entry leaves the cache. At each cell cycle, a DC copies the current signals from the signal matrix and calculates the temporary output cytokines. The cycle rate and the signal weights are parame-

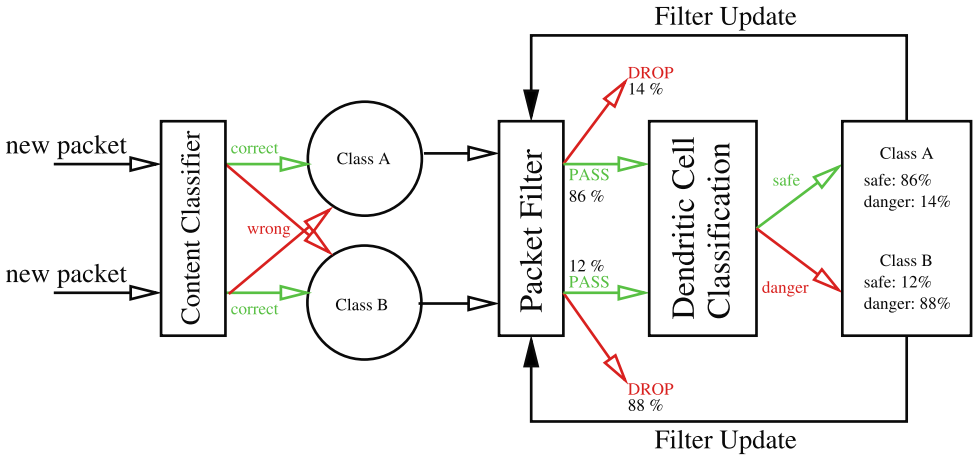


Fig. 8 Behaviour and content classification

Table 1 Summary of the design choices

Design choice	Design decision
total number of DCs	as many as cache entries
antigen capacity	1
multiple sampling	no
maturation time	when the captured entry is removed from cache
cycle rate	variable (see experiments)
signals weights	variable (see experiments)

ters that have to be evaluated later. All design choices are summarised in Table 1.

The packet classification by a Dendritic Cell can be interpreted as a behaviour classification. A packet enters the system, more precisely the interest cache, and its behaviour causes the release of various signals. A DC captures this packet and collects signals from the signal matrix. When the packet leaves the cache, the DC classifies it as either the cause of malicious or safe behaviour. Both behaviours are clearly not only caused from one single interest packet captured by a DC. For example, when an attack starts, danger and PAMP signals are released. Most likely, there will be still benign interest messages in the cache at this time and they are possibly misclassified as the cause of malicious behaviour. However, in order to maintain a packet filter, the IDS needs to identify each packet as either malicious or safe.

As the classified packet will not be sent again, the next task of the system is to extract information that reflects safe and dangerous packets from the packets classified by the DCs. This information is then used to configure the packet filter that drops malicious packets before they can cause damage. In the human immune system, this automatic response is triggered by immune cells such as T-cells. In our system, the response will be handled by the second stage of the algorithm, which is described in more detail in the next section.

5.2.6 Content classifier and packet filter

The aim of the content classifier and the packet filter is to trigger an automated response to the interest cache poisoning attack. Bogus packets should be filtered before they enter the interest cache and cause damage. The packet filter needs the information whether a new packet is dangerous or safe in order to make the correct filter decision. The system needs to extract generalised patterns from the packets that have been classified by a DC. These patterns should reflect features that distinguish safe and dangerous packets.

To extract these patterns, a content classifier is used. The algorithm to be used for the content classifier is a kind of clustering algorithm. Figure 8 shows a simple example. A new interest packet is classified by the content classifier and partitioned into class A or B. The packet filter initially does not drop any packet and thus the packet is stored in the interest cache and is classified by a DC as either safe or dangerous.

The information about the class (A or B) and the information about the DC decision (safe or dangerous) are then used to update the filter. The filter maintains a drop rate r_{dropA} for packets partitioned into class A and r_{dropB} for packets partitioned into class B. The drop rate reflects the history of the last N packets of each class. For example, if 86 of the last 100 packets of class B have been classified as dangerous, the drop rate r_{dropB} is 86%. Hence, a new interest packet that is classified as belonging to class B is dropped by the packet filter with a probability of 86%. Each packet that passes the filter causes the update of the filter drop rates according to its class and its DC classification.

Design decisions regarding the content classifier and the packet filter are the length of the decision history and the number of different classes. The length of the history determines the effect of a single packet on the drop rate. Using a shorter history, a single packet has a greater influence on the drop rate. If the number of classes is not predefined, the content classifier can dynamically create a new class if the

content of the packets differs too much from the existing classes.

Another issue is that if many packets from one class have been classified as dangerous, the drop rate for this class will be very high. Hence, fewer packets will enter the system and be classified by the Dendritic Cells. As a result, the drop rate for that class cannot change back to normal very easily. This makes it hard to normalise the drop rate if the initial DC classifications were wrong for this class. Thus, a filter has to age its decision in some way; for example, the drop rate can be decreased every time interval. Another possibility is to decrease the drop rate if less danger and PAMP signals are released. Yet another option is to reset the filter completely after a certain time. The length of the decision history and the reset rule of the filter are evaluated in [15].

As this work focuses on the detection part of the algorithm, the second stage of the algorithm is kept simple and is only simulated. Nevertheless, the response part of the algorithm is important and needs to be evaluated. With the assumption that packets sent by the sink node and packets sent by the attacker have a different content, the number of classes is simply 2. The accuracy of the content classifier is simulated by setting a probability $P_{accuracy}$ that defines the accuracy of the classifier. For example, $P_{accuracy} = 0.8$ means that the classifier partitions 80% of the packets into the correct class. Therefore, when our simulation sets $P_{accuracy} = 0.8$, we assume that the content classifier performs with the accuracy 80%. Simulating the content classifier allows us to investigate the performance of the automatic response.

6 Experiments

Three sets of experiments were carried out on SNAIS in order to evaluate its behaviour. The first was an assessment of the impact of the new cache poisoning attack proposed in this work. The TinyOS implementation was used (for reasons of speed) to assess four variations of the attack. The second set of experiments assessed the detection capabilities of both implementations of SNAIS and their sensitivity to different internal parameter settings. The final set of experiments assessed the effectiveness of both the detection and response of Sim-SNAIS and Tiny-SNAIS when under attack.

6.1 The impact of the cache poisoning attack

This first section measures the impact of the interest cache poisoning attack. Four different versions of the attack are performed and then compared in terms of effectiveness and efficiency.

6.1.1 *Effectiveness measurements*

The effectiveness measurement is performed to measure the impact and success of the attack. Since the goal of the attacker is to suppress the data flow from the sensor nodes to the sink node, the success of the attack can be measured by counting the number of data packets received by the sink node. N_{normal} is the number of data packets received without the presence of the attacker and N_{attack} is the number of data packets received with an attacker mounting the cache poisoning attack. The experimentally determined suppression rate $r_{suppress,exp}$ is calculated as:

$$r_{suppress,exp} = 1 - \left(\frac{N_{attack}}{N_{normal}}\right).$$

For each run N_{normal} and N_{attack} are measured and $r_{suppress,exp}$ is derived. Additionally the mean and standard deviation for both counters and for the suppression rate are calculated.

6.1.2 *Efficiency measurements*

The efficiency measurement compares the attack variations in terms of resources used while performing the attack. Clearly, an attacker wants to cause as much damage as possible, but since a malicious node has the same energy resources as a normal node, the attacker does not want to waste his energy. The attack cannot be continued if the malicious node has no more energy to send out bogus packets. For nodes in sensor networks most energy is consumed when performing sending and receiving operations. Thus, we use B, the number of bogus packets sent out by the attacker, as a metric of resources used for the attack. The effectiveness metric defined in Sect. 6.1.1 is combined with the energy used, in calculating an efficiency index. With the assumption that the sink requests the same data rate from each sensor, the theoretical best case suppression rate $r_{suppress,theo}$ can be expressed as follows:

$$r_{suppress,theo} = 1 - \left(\frac{C - B}{S}\right); \quad C - B > 0$$

where C is cache size, B is number of bogus packets sent out, S is number of subscriptions (with each subscription resulting in data delivered).

For example, with $S = 2$ subscriptions and a cache size of 10, the attacker needs to send 10 bogus packets to achieve a rate of 100% and 9 packets for 50%.

For the length of the experiment, the sink sends out the interests several times. If I is the refresh interval and T is the run time of the experiment, the number of updates is $n = T/I$. Given the experimental suppression rate $r_{suppress,exp}$

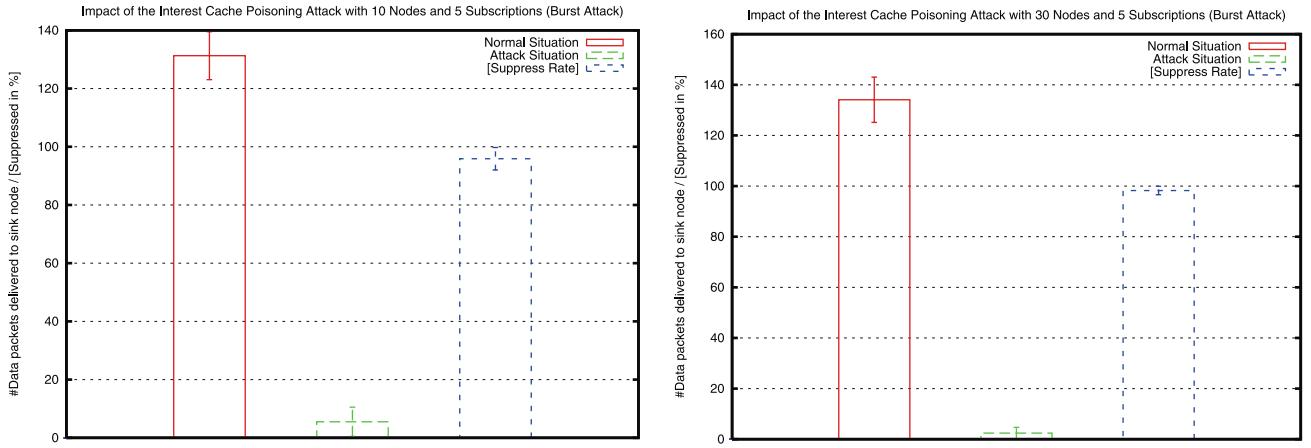


Fig. 9 Effectiveness of the burst attack (10 nodes *left*, 30 nodes *right*)

Table 2 Overview of the fixed parameters for this section

Parameter	Value
nodes	10 [30]
topologies	10
run time	each 300 seconds
sink subscriptions	5
data interval	10 seconds
refresh interval	60 seconds
interest expiration	75 seconds

from the experiment, the following formula gives the theoretical number of bogus packets needed to achieve this suppression rate:

$$B_{needed} = n \cdot (S \cdot (r_{suppress,exp} - 1) + C).$$

The theoretical suppression rate is compared to the experimental rate. Therefore, the efficiency of the attack is defined as

$$E_{attack} = \frac{B_{needed}}{B_{sent}}.$$

6.1.3 Experimental settings

Ten different network topologies were generated [15]. For each topology a normal run and a run for each different attack was performed. For each run, the number of data packets delivered to the sink was measured. The mean number of packets was calculated and used to show and compare the effectiveness and efficiency of the attacks. Four different types of the poisoning attack are compared. The attacks and the reason why these four types were chosen are explained in the following sections. Table 2 gives an overview of the fixed experimental parameters.

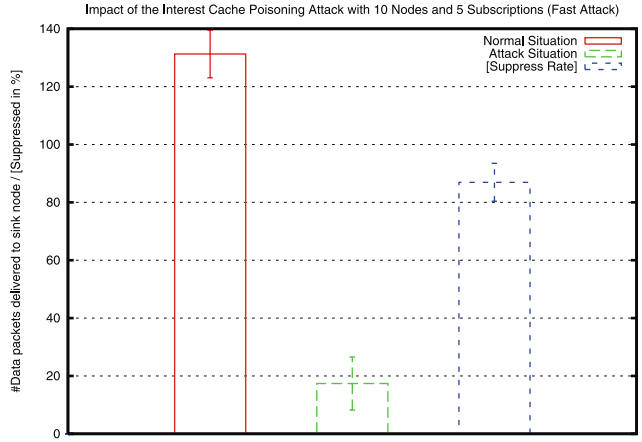
6.1.4 The burst attack

The first experiment shows the impact for the burst attack as described in Sect. 4.3.2. C packets are sent out every time the attacker receives a new interest, where C is the size of the interest cache. The attacker does not send out bogus packets for duplicated benign interests. Since the attacker stores the benign interests in its interest cache he is able to tell if an interest is duplicated or new. This attack is easy to perform but, on the other hand, a waste of energy, because a lot of unnecessary packets are sent out. The simple attack should show the concept of the cache poisoning attack. Figure 9 (left) shows the results of the experiment using 10 nodes. The mean number of data packets delivered to the sink is 131.3 during a normal situation. The same run with an attacker sending C bogus packets for each received interest results in only 5.5 received packets on average. This is a suppression rate of nearly 96%. So the effectiveness of this burst attack is very good. The result for the attack with 30 nodes is shown in Fig. 9 (right). With 7.2 received packets on average, the attack is still very good.

6.1.5 The fast attack

The fast attack requires a smarter attacker than the burst attack. An observation of the network behaviour shows that the sink node sends out the interest packets for its subscriptions as a bundle. Therefore, the attacker can figure out when the last interest packet for a refresh period has arrived and then starts the attack. This saves the attacker a lot of packets (and energy) and should have the same suppression rate.

The results for the fast attack are still very good. The mean suppression rate is 87% with 17.4 data packets delivered to the sink on average, as shown in Fig. 10 (left). With 5 sink subscriptions, this attack needs only 20% of the amount of packets that the simple burst attack needs. The results for the attack in a network with 30 nodes are shown



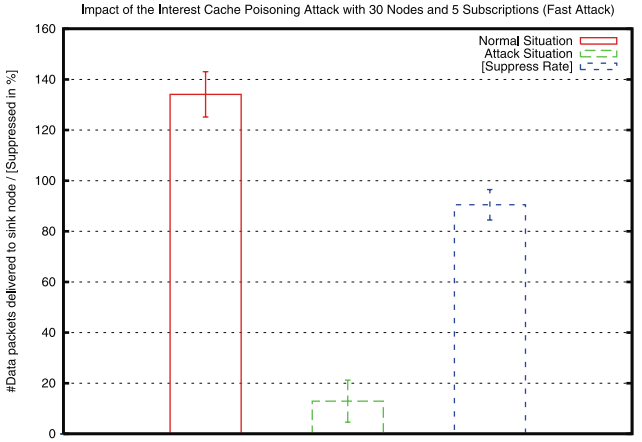
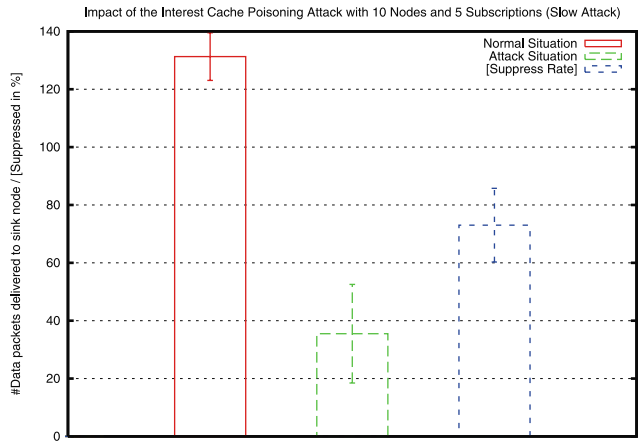


Fig. 10 Effectiveness of the fast attack (10 nodes *left*, 30 nodes *right*)



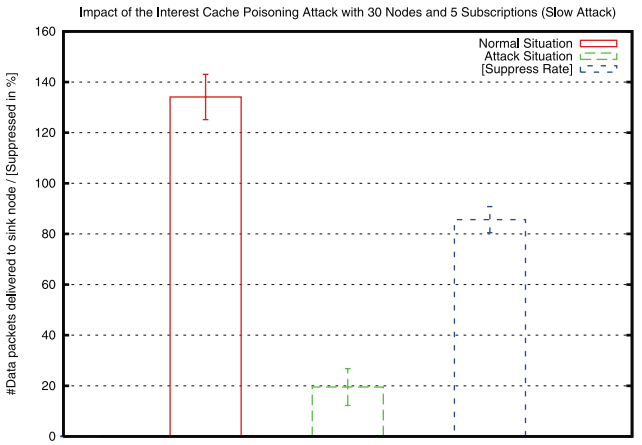


Fig. 11 Effectiveness of the slow attack (10 nodes *left*, 30 nodes *right*)

in Fig. 10 (right) and do not differ much from the results for 10 nodes.

6.1.6 The slow attack

An attacker that is aware of burst detection will try to behave more like any other node during the attack. Therefore, he sends out the packets with the speed the Tiny Diffusion implementation suggests. That means that there is a randomly chosen pause between two consecutive bogus packets. The pause lies in the range between 0.125 and 1 second. The attacker also waits for a sink node to send all interest packets before starting the attack.

The suppression rate of this attack goes down to 73% with 35.5 data packets delivered to the sink on average. The number of packets used is the same as in the fast attack. Therefore, the effectiveness of the slow attack is lower than the fast one. The results for a network with 30 nodes are shown in Fig. 11 (right). The results are better than for 10 nodes (Fig. 11, left). One reason for that could be the higher density of the larger network. In the topologies generated with 30 nodes, the nodes have more neighbours which leads to more duplicated packets. Full message buffers might result in data packets being dropped.

6.1.7 Comparison of the efficiency

The attacks are now compared using the efficiency metric described in Sect. 6.1.2. Figure 12 gives an overview of the results for 10 and 30 nodes using the mean values. Clearly, the burst attack is inefficient, but the effectiveness is the best. The differences between the two other attacks are in terms of efficiency, but the fast attack is more effective than the slow one.

6.1.8 The constant attack

The constant attack is performed by sending out bogus interest packets constantly. The attacker does not calculate the correct time to start the attack. Every n seconds a bogus interest is sent out. We compare different attacks by varying

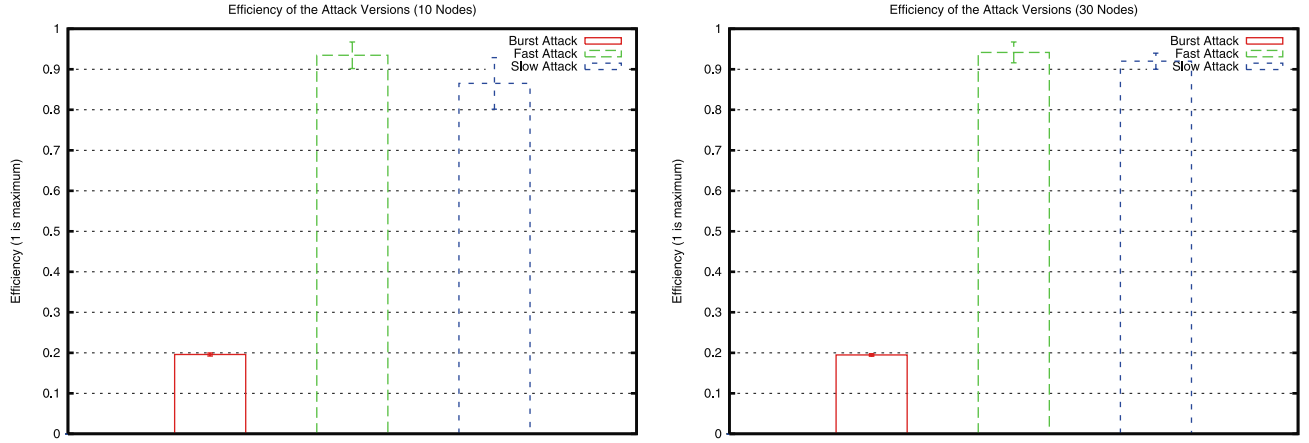


Fig. 12 Efficiency of the different attack versions

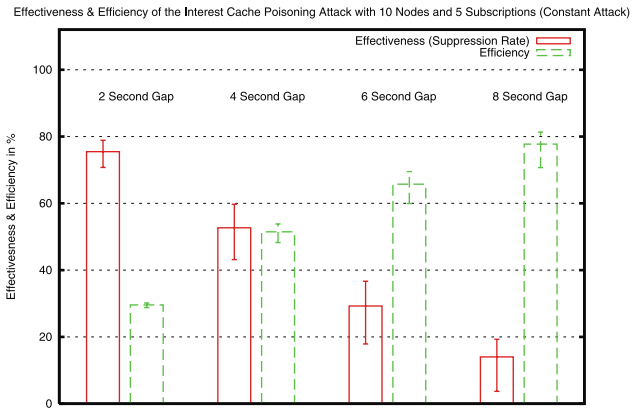


Fig. 13 Effectiveness of the constant attack (10 nodes)

the pause between two consecutive bogus packets. The simulation time is now 350 seconds and not 300 seconds as for the previous experiments. This is because the constant attack is used in Sect. 6.2 to measure the detection results with a different attack pattern and was originally not supposed to appear in this section. Due to time constraints, we only evaluate this attack with a network of 10 nodes. Figure 13 shows an overview of the effectiveness and the efficiency of the constant attack. Overall, the constant attack shows worse suppression results than the previous attacks. Clearly, with a larger gap between two consecutive packets, the effectiveness of the attack becomes worse, although the efficiency is better.

Regarding the number of bogus packets that are sent out, the constant attack with a pause of 6 seconds between two consecutive packets can be compared with the previous attacks, because it sends out about the same number of bogus packets. The achieved suppression rate of the constant attack is much worse than the rate of a burst, fast or slow attack. The efficiency of the constant attack is worse than the efficiency of the fast and slow attack, but better than the efficiency of the burst attack.

6.2 Evaluation of the detection mechanism

This section evaluates the accuracy of the Dendritic Cell Algorithm within SNAIS and assesses its sensitivity to changes in signal weights and other parameters.

6.2.1 Experimental setup

Table 3 shows the 4 different sets of weights used for the six signals described previously. The first signal weight set contains the weights suggested by Greensmith *et al.* [3]. Set 2 and 3 increase the weights of the danger and PAMP signals, because the first set showed too many false negatives (shown as 'bogus wrong' in the figure). Table 4 gives the main fixed parameters for the two implementations of SNAIS. (Note that the number of nodes was 7 in Sim-SNAIS in order to make the simulation time practical.) Finally, Table 5 shows the variables changed in each of the six experiments.

6.2.2 Results

The detection results are shown in Fig. 14. Overall, the intrusion detection results show hardly any false positives (benign wrong) for Tiny-SNAIS and few wrong for Sim-SNAIS. The signal weight set 3 shows the best results for both implementations. More bogus packets are classified correctly.

Comparing experiments 1–3 with 4–6, for Tiny-SNAIS the simple matrix shows better results than the large matrix. This can be explained with the fact that the simple matrix models the biological mechanism better, because it reflects the classification of antigens within the context. In other words, even bogus packets that do not cause the release of danger signals are still classified as dangerous, because the

Table 3 Different signal weight sets used for the experiments

Weight set	Cytokine	PS	DS1	DS2	SS1	SS2	SS3
1	Semi-Mature	0	0	0	3	3	3
	Mature	2	1	1	−3	−3	−3
2	Semi-Mature	0	0	0	2	2	2
	Mature	6	4	4	−1	−1	−1
3	Semi-Mature	0	0	0	2	2	2
	Mature	12	8	8	−1	−1	−1

Table 4 Fixed parameters for the experiments

Parameter	Tiny-SNAIS value	Sim-SNAIS value
Node number	10	7
Different topologies (runs)	10	1
Simulation time per run	350 seconds	300 seconds
Maximum signal age	10 seconds	10 seconds
Sliding window size (DS1/SS3)	10 seconds	10 seconds
Sliding window shift size	1 second	1 second
Number of subscriptions	7	5
Interest expiration	75 seconds	75 seconds
Interest cache size	10	10
Refresh time	60 seconds	60 seconds
Attack mode	Fast mode	Fast mode
Bogus packets per attack	10	10
DC cycle rate	0.5 seconds	0.5 seconds
Update rule	Simple (overwrite)	Simple (overwrite)

Table 5 Parameters varied for the experiments

Experiment	Parameter	Value	Parameter	Value
1	Signal weight set	1	Matrix structure	Simple
2	Signal weight set	2	Matrix structure	Simple
3	Signal weight set	3	Matrix structure	Simple
4	Signal weight set	1	Matrix structure	Large
5	Signal weight set	2	Matrix structure	Large
6	Signal weight set	3	Matrix structure	Large

DCs that capture these bogus packets still receive danger and PAMP signals caused by the other bogus packets. For example, a bogus packet that is inserted in an empty cache slot does not cause the release of danger signal DS2, but other bogus packets that overwrite the entries do. Hence, the context is rather dangerous than safe. This is not the case with the large signal matrix where packets are mainly classified according to the signals they cause.

While the Tiny-SNAIS results show that the simple matrix always performed better than the large matrix, the results shown for Sim-SNAIS do not indicate the simple matrix is always better. In fact, comparisons show that there is not much difference in true positive (TP) and false positive (FP) rates and the best true negative (TN) rates in the third signal weight set is not significantly different. However, it should be noted that the large matrix used in Sim-SNAIS is different from the large matrix used in Tiny-SNAIS (see Sect. 5.2.3). The matrix of Sim-SNAIS was designed so that it can hold all signals as long as the signals do not get old.

With this difference, it can be observed that the best TN rates are shown to be at a similar level, which implies that there is no great difference between interpreting the context with only the most recent signal and interpreting the context with the average signal vote results that were collected for the previous 60 seconds.

6.2.3 Analysis

It is clear that the different signal weights do have great impact on the DC's detection rates. These results are consistent in both implementations of SNAIS. The relatively good results in TP and FP rates do not require changing the signal weights in the semi-output cytokines. However, the poor results in TN and FN rates led us to create new sets of signal weights that reduce the safe signal weights by one third and double and triple the danger and PAMP signals respectively in calculating the mature output cytokines. The results

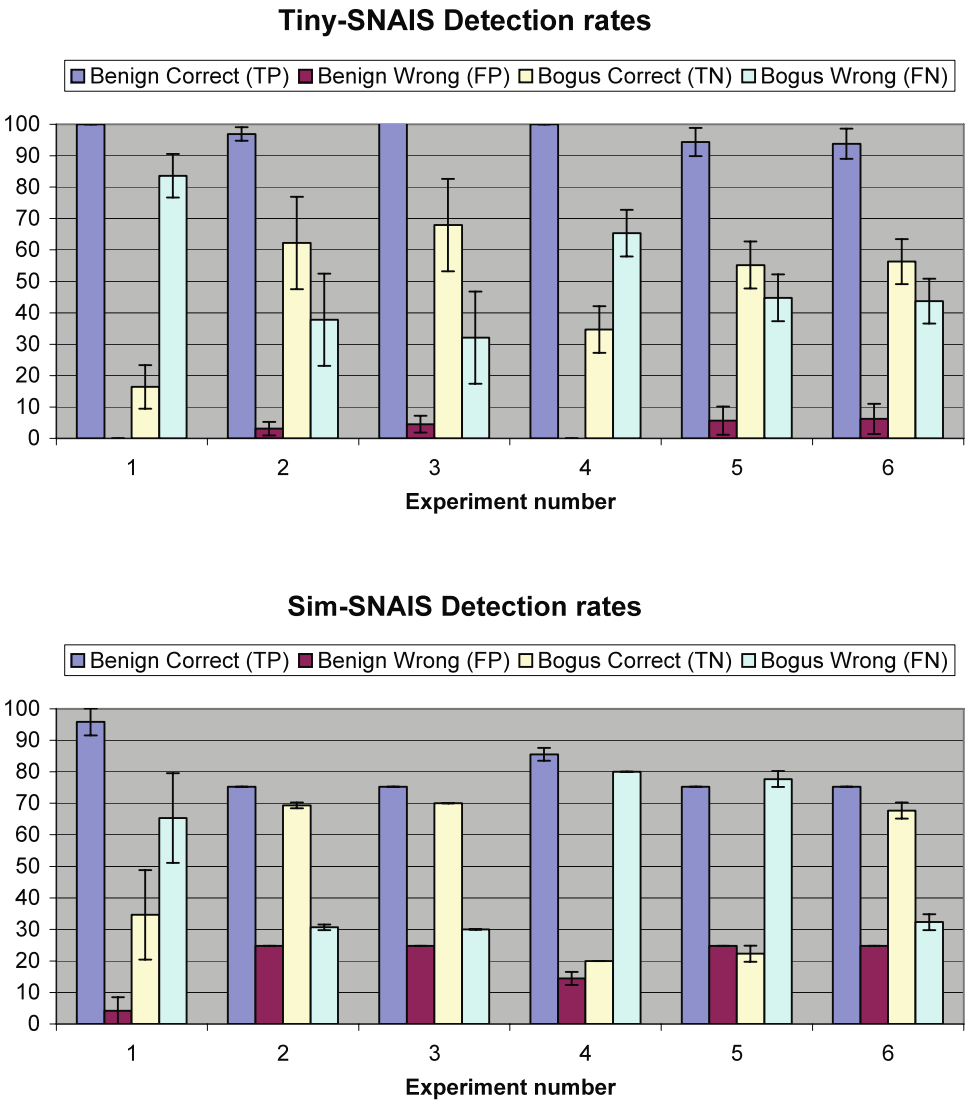


Fig. 14 Detection rates for Tiny-SNAIS and Sim-SNAIS in experiments 1–6

show that the reduction of safe signal weights dropped the TP rates around 10% while it increases the TN rates greatly (max nearly 50%). On the other hand, without the safe signal weight decrease, the increase of danger and PAMP signals increase the TN rate while the TP rate no longer drops significantly. Further increase of the danger and PAMP signal weights no longer increases the TN rate largely although it has shown some increase.

The above results clearly suggest that the appropriate setting of signal weight values is crucial for the DCA to work properly. Although the initial signal weights were borrowed from actual biological statistics, these are specific statistics and they do not necessarily conform to different computer applications. Therefore, a method for automatically setting the signal weights would be useful in the future. One interesting finding from these experiments is that the initial results of signal weight setting can provide some clues to adjust the weights. Although different results would be expected depending on the applied application areas, it appears likely that the DCA might be able to adapt its signal weights automatically depending on the initial detection results it is producing. As shown in these experiments, if a high TP rate and low TN rate are produced in early results, the DCA can simply start to increase the danger signal or PAMP signal weights.

6.2.4 Number of sink subscriptions

Having explored the effects of different signal weights and matrices, the following experiments investigate how a the number of interest subscriptions affects the detection accuracy of the intrusion detection system. We expect SNAIS to perform better when the sink node subscribes to more data streams. This is because an interest cache poisoning attack causes more damage. Therefore, more danger and PAMP

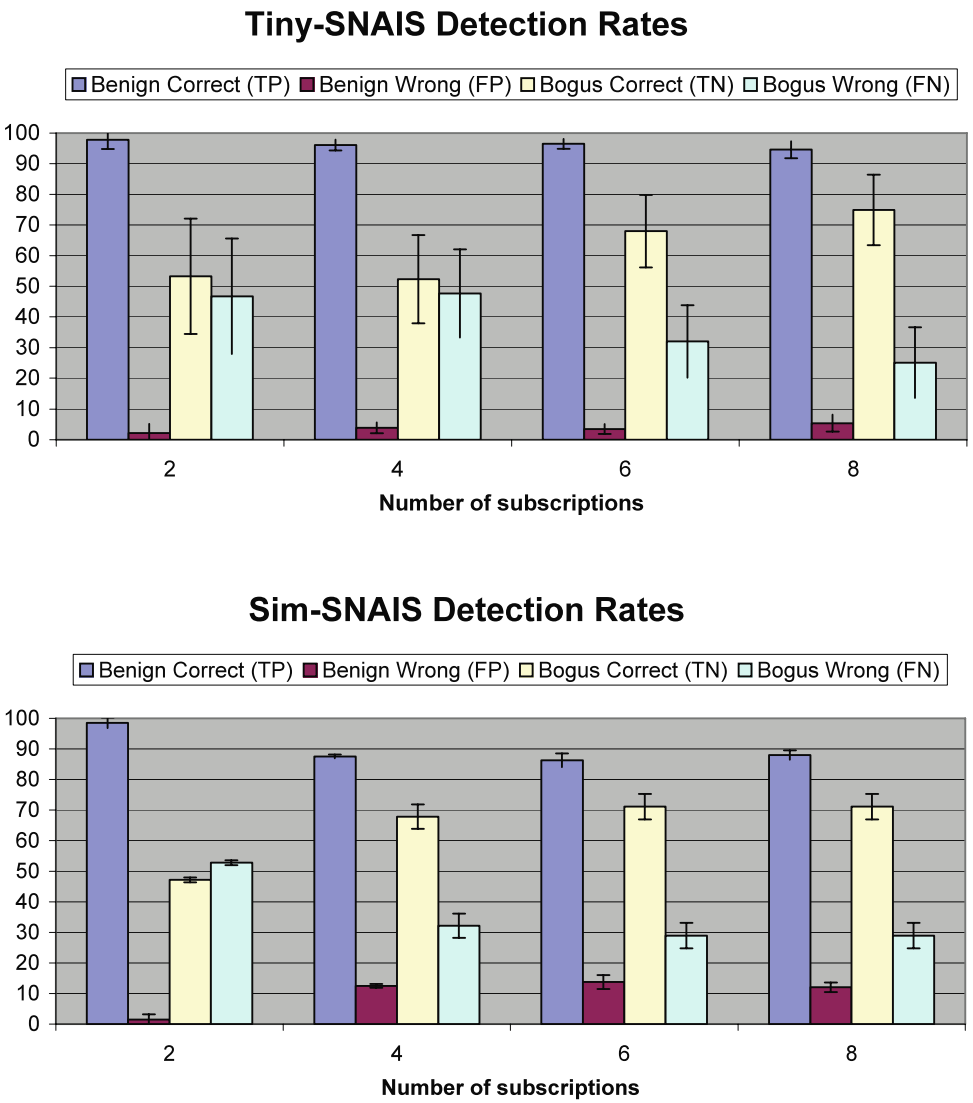


Fig. 15 Detection rates for Tiny-SNAIS and Sim-SNAIS in the sink subscription experiments

signals are generated, leading to better detection results. For the experiments the fixed parameters shown in Table 4 are used again and only the number of interest subscriptions is varied. The third signal weight set and the simple signal matrix are used for these experiments as they showed the best results in the previous experiments. 10 runs for each experiment with 2, 4, 6 and 8 subscriptions were performed on both implementations of SNAIS.

Figure 15 plots the simulation results. It can be observed that the number of correctly classified bogus packets is affected by the number of subscriptions, while the number of false positives does not change much. In Tiny-SNAIS there is no significant difference between two and four subscriptions; in Sim-SNAIS there is a 10% drop in TP rates when the number of subscriptions is increased from 2 to 4. Both implementations show that the overall detection results become better (improved TN rates) with a higher number of subscriptions.

6.3 DCA + response test

6.3.1 Experiment objectives and setup

This section evaluates how the automatic response affects the impact of the interest cache poisoning attack. First, a content classifier with a classification accuracy of 80% and two classes A and B was tested.

In this setting, we assume that class A indicates 'benign' packets and class B represents 'bogus' packets. This means that an interest packet, which has been sent by the sink node, will be correctly classified as class A with a probability of 80%. A packet that has been sent by the attacker will be classified as class B with a probability of 80%. However, with an error rate of 20%, the content classifier classifies 'benign' packets as class B and 'bogus' packets as class A respectively. The packets classified by the content classifier are passed to the packet filter. Then, the packet filter drops inter-

est packets according to the packet pass rate of each class, which is constantly updated by the DC decisions. For instance, if the pass rate of class B is currently 75%, an interest packet that has been classified as class B is dropped with a probability of 25%.

Second, SNAIS was tested without a content classifier. Using this setting, newly arrived packets are no longer classified by a content classifier, they are directly handled by the packet filter. The packet filter maintains only one pass rate, which is configured by the number of safe and dangerous packets classified by the DCs within the last n packets. For example, if 10 of the last 30 packets have been classified as dangerous, the pass rate of the filter is set to 66% and packets are dropped with a probability of 33%.

We expect the packet filter with the content classifier to perform better than the one without content classifier, because it will drop fewer benign interests. However, it is not clear whether a content classifier can be deployed on a sensor node and how accurately it can classify the packets. Moreover, a very clever attacker can try to make the benign packets look dangerous. As a result, the packet filter would drop benign packets and the data delivery to the sink node will be disturbed. The setting without the content classifier is not vulnerable to this attack. Although the content classifier is suggested by the inspiration of immune cells, the limited capacity of sensor nodes might prevent the implementation of this idea. Hence, we test this alternative but simpler option in the second set of experiments.

The automatic response was evaluated by using a sliding window to measure the number of data packets that have been received by the sink node within the last 60 seconds. This measurement was done for the normal situation without the presence of an attacker and for both different packet filter configurations. Furthermore, the current packet pass rates were recorded and the mean rates were calculated among all the nodes. We use a simulation time of 1000 seconds, which is longer than the time in the previous experiments. This is because the response mechanism needs some time before it can react to the attack. A bit field with a length of 32 bits is used to keep track of the number of safe and dangerous packets. Both implementations of SNAIS use the settings that showed the best results in Sect. 6.2 and the fixed parameters are the same as shown in Table 4.

6.3.2 Results

Figure 16 shows the results of both filter configurations, showing packet pass rates of each class and the number of data packets delivered in 3 different cases: (i) normal situation without an attacker present, (ii) attack without the packet filter active and (iii) attack with the packet filter active. The average packet pass rates are calculated among all nodes.

Examining the results for Tiny-SNAIS, initially no bogus packets are dropped by the filter, and not many data packets are delivered to the sink node. As a result of the attack, danger and PAMP signals are released and packets start to be classified as dangerous. Thus, the pass rates of the packet filters decrease. As more interest packets are dropped, the attack is less successful, and hence more data packets are routed back to the sink node. This has the effect that the concentration of the danger and PAMP signals becomes weaker. As a consequence, fewer interest packets are classified as dangerous and the packet pass rates increase again. More bogus packets are inserted in the interest caches again and the impact of the attack is higher. Afterwards, the whole process is repeated.

The correlation between the pass rates and the number of delivered data packets can be clearly seen. Overall, the results of the packet filter without the content classifier do not differ much from the results of the packet filter with the content classifier. This is a promising result as the filter without the content classifier is a much simpler response mechanism. However, the filter with content classifier cannot do worse than the filter without the content classifier because it uses additional information to make its filter decision.

Unexpectedly, the results for Sim-SNAIS differ slightly to those for Tiny-SNAIS. As described, for Tiny-SNAIS, the average bogus packet pass rate decreased until some implicit threshold is reached and then increased again. However, the results for Sim-SNAIS show that the bogus packet pass rate continuously decreased until it reached zero and never increased again. Further investigation reveals that the difference in results may originate from the number of nodes used in each network. In the Sim-SNAIS experiments, the network size is very small and most of nodes become the neighbour nodes of the attacker's node. Even the furthest node from the attacker's node is only two hops away and so similar trends can be observed. Nevertheless, despite this difference, the overall result for Sim-SNAIS using the packet filter without the content classifier conforms to the results seen for Tiny-SNAIS.

7 Conclusions

In this work we described SNAIS (Sensor Network Artificial Immune System)—an IDS approach for Directed Diffusion based sensor networks inspired by the Danger Theory and the Dendritic Cell Algorithm [4]. This work also introduced a new attack on sensor networks employing the Directed Diffusion routing protocol. The attack highlights a general vulnerability in sensor network protocols that rely on caches with limited capacities to keep track of state of the network.

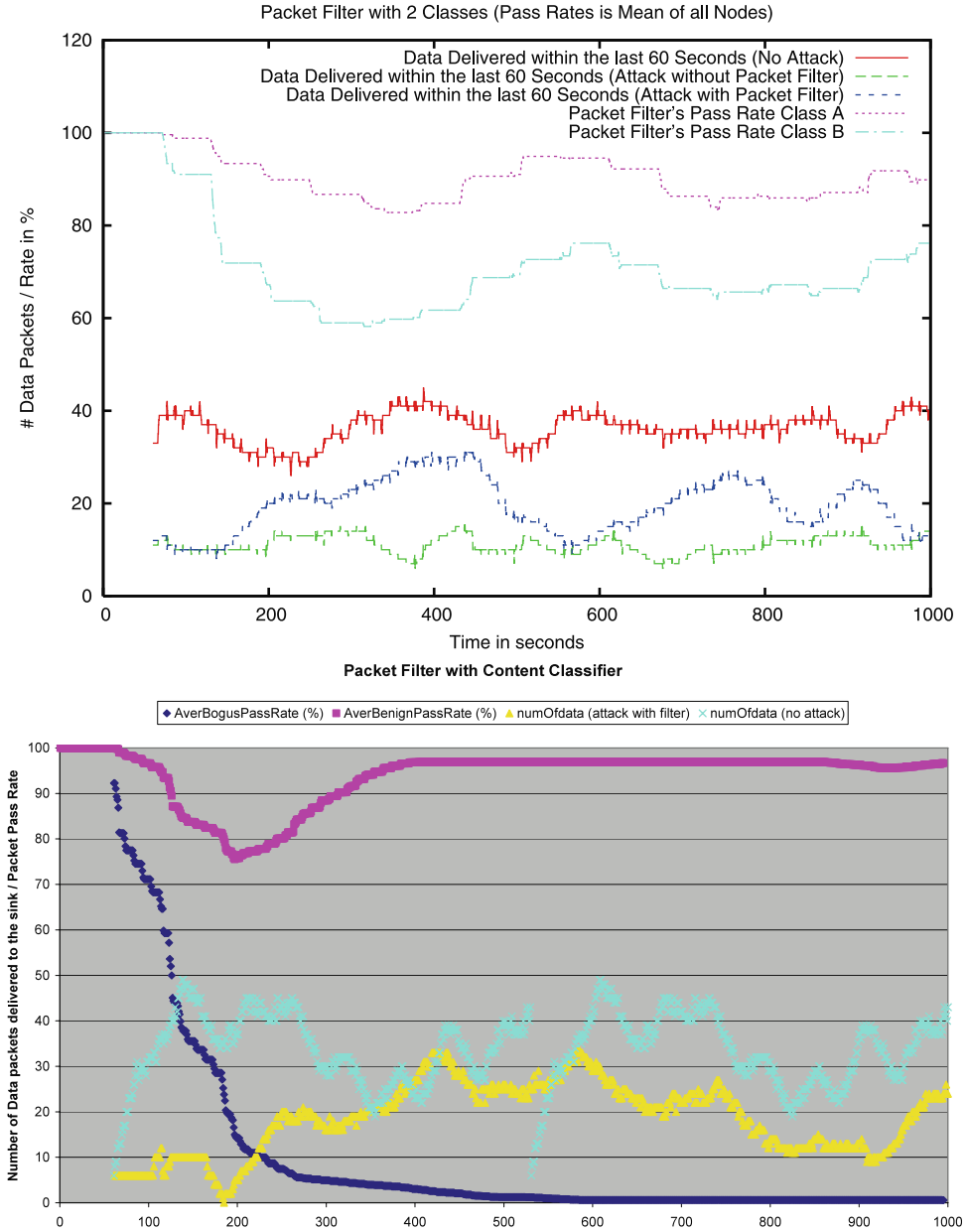


Fig. 16 Packet pass rates of each class and the number of data packets delivered, with and without content classifier (Tiny-SNAIS results shown with a clear background, Sim-SNAIS results shown with a grey background)

Two implementations of SNAIS were evaluated: one hardware specific for the *T-mote Sky sensor* using TinyOS, and one simulation (enabling the full use of the Direct Diffusion protocol) using J-Sim. The experiments have shown that a Danger Theory inspired approach shows promising detection results.

In more detail, the evaluation of the interest cache poisoning attack showed that the attack heavily affects the successful operation of a sensor network that employs the Directed Diffusion routing protocol. It can easily disturb the data delivery from the sensor nodes to the sink node. The investigation of different signal weights showed that SNAIS is sensitive to the weight values, but that these values can be fine-tuned relatively easily, which implies that this process could be automated in the future. Both implementations of SNAIS were shown to be capable of detecting instances of the interest cache poisoning attack.

A packet filter, which implements the automatic response of the intrusion detection, was also briefly evaluated. Using the packet filter in combination with the context classifier and without a content classifier showed acceptable results. Future work should analyse whether a content classifier can

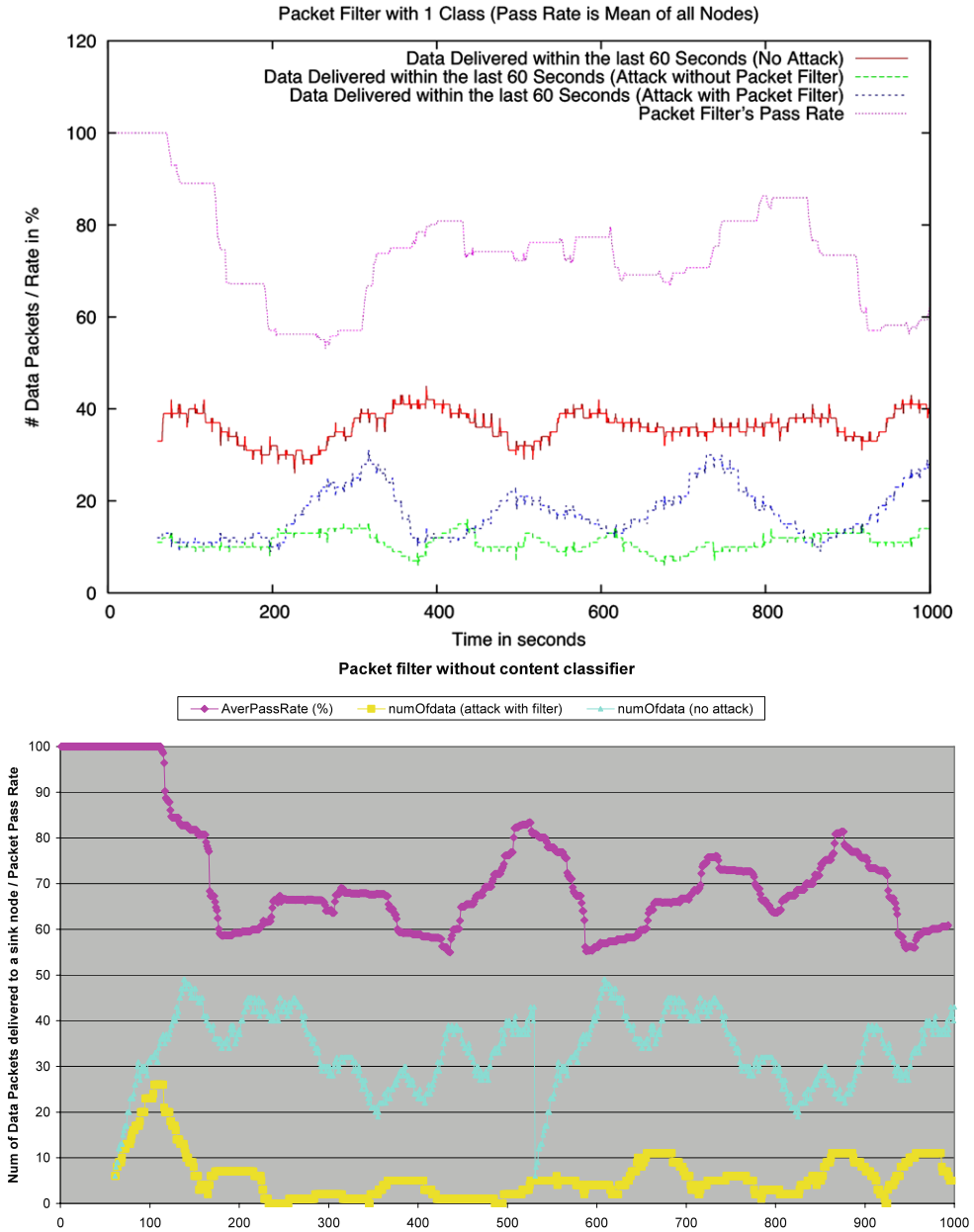


Fig. 16 (*Continued*)

be deployed on a sensor node at all and how accurately it is able to classify the packets. However, the packet filter without the content classifier, which is a much simpler solution showed promising results.

Sensor and other similar dynamic networks are becoming increasingly common. It is important to realise that attacks that exploit the weaknesses in their hardware and software will also become common, as they have become for desktop computers on the Internet. This work has suggested one possible form of attack. It has also shown that a lightweight detection system using an analogy of the immune system by exploiting signals derived from system behaviours is useful and effective.

Acknowledgements Thanks to the members of the 1st RUNES project (http://www.ist-runes.org/) at UCL for providing the equipment used in this project. This work was supported by EPSRC (GR/S47809/01).

References

1. Matzinger P (1994) Tolerance, danger and the extended family. Annu Rev Immunol 12:991–1045
2. Aickelin U., Bentley P., Cayzer S, Kim J, McLeod J (2003) Danger theory: the link between AIS and IDS. In: Proceedings of the 2nd international conference on AIS (ICARIS-03), pp 147–155

3. Greensmith J, Aickelin U, Cayzer S (2005) Introducing dendritic cells as a novel immune-inspired algorithm for intrusion detection. In: Proceedings of ICARIS-05. Springer, Berlin, pp 153–167
4. Greensmith J, Twycross J, Aickelin U (2006) Dendritic cells for anomaly detection. In: Proceedings of IEEE congress on evolutionary computation (CEC-06), Vancouver, Canada
5. Bentley P, Greensmith J, Ujjin S (2005) Two ways to grow tissue for AIS. In: Proceedings of the 3rd international conference on AIS (ICARIS-05). Springer, Berlin, pp 139–152
6. Kim J, Wilson W, Aickelin U, McLeod J (2005) Cooperative automated worm response and detection ImmuNe algorithm (CARDINAL) inspired by T-cell immunity and tolerance. In: Proceedings of the 3rd international conference on AIS (ICARIS-05), pp 168–181
7. Twycross J, Aickelin U (2005) Towards a conceptual framework for innate immunity. In: Proceedings of the 3rd international conference on AIS (ICARIS-05). Springer, Berlin, pp 153–167
8. Twycross J, Aickelin U (2006) LIBTISSUE—implementing innate immunity. In: Proceedings of the CEC-06, Vancouver, Canada
9. Sarafijanovic S, Le Boudec J (2005) An AIS for misbehaviour detection in mobile ad-hoc networks with virtual thymus, clustering, danger signals and memory detectors. In: Proceedings of the 2nd international conference on AIS (ICARIS-04). Springer, Berlin, pp 342–356
10. Greensmith J et al (2006) Articulation and clarification of the dendritic cell algorithm. In: Proceedings of ICARIS-2006
11. Intanagonwiwat C et al (2003) Directed diffusion for wireless sensor networking. IEEE/ACM Trans Netw 11(1):2–16
12. Akyildiz IF et al (2002) A survey on sensor networks. IEEE Commun Mag August:102–114
13. Estrin D, Cullar D, Pister K, Sukhatme G (2002) Connecting the physical world with pervasive networks. In: Pervasive computing, pp 59–69
14. Karlof C, Wagner D (2004) Secure routing in wireless sensor networks: attacks and countermeasures. In: Ad hoc networks, pp 293–315
15. Wallenta C (2006) Detecting malicious activities in directed diffusion based sensor networks. Diploma thesis, System Architecture Group, Department of Computer Science, Universität Karlsruhe
16. Mati S, Giuli T, Lai K, Baker M (2000) Mitigating routing misbehaviour in mobile ad hoc networks
17. Lioy A, Maino F, Marian M, Mazzocchi D (2000) In: DNS security, TERENA networking conference

Christian Wallenta gained a MSc in Computer Science from Universität Karlsruhe (TH) in September 2006. During his master's dissertation he worked at University College London (UCL) as a visiting researcher before moving to Oxford to start his DPhil. His master's dissertation is entitled "Detecting Malicious Activities in Directed Diffusion Based Sensor Networks". During his studies he worked part-time for Consileon Business Consultancy GmbH, Karlsruhe, Germany.

Jungwon Kim research interests are applying AI or statistical techniques to solve real-world problems. She is especially interested in pattern discovery from huge volumes of messy data using adaptive AI techniques such as genetic algorithms (evolutionary computing), computer immunology, neural networks and other algorithms. Her recent research interest was focused on the understanding the human immune mechanism in order to apply it to develop a novel intrusion and fraud detection system. She obtained her PhD in this area at UCL in 2002, and continued her research on the topic in a Postdoc position at Kings College followed by a second Postdoc position back at UCL.

Peter J. Bentley is an Honorary Senior Research Fellow at the Department of Computer Science, University College London (UCL), Collaborating Professor at the Korean Advanced Institute for Science and Technology (KAIST), a Visiting Research Fellow of the University of Kent and a freelance writer. Peter runs the Digital Biology Interest Group at UCL. His research investigates evolutionary algorithms, computational development, artificial immune systems, swarming systems and other complex systems, applied to diverse applications including design, control, novel robotics, nanotechnology, fraud detection, security, art, and music composition. Peter was nominated for the $100,000 Edge of Computation Prize in 2005. He regularly gives plenary speeches at international conferences and is a consultant, convenor, chair and reviewer for workshops, conferences, journals and books in the field of evolutionary computation and complex systems. He has published over 160 scientific papers and is editor of the books "Evolutionary Design by Computers", "Creative Evolutionary Systems" and "On Growth, Form and Computers", and author of "The PhD Application Handbook" and the popular science books "Digital Biology", "The Book of Numbers" and "The Undercover Scientist".

Stephen Hailes was an undergraduate at Trinity College Cambridge and then a PhD student in the Cambridge University Computer Lab, working in the field of distributed object-based programming languages. Today he is Deputy Head of Department and Director of Studies at the Department of Computer Science, University College London. His research interests lie in the fields of mobile systems and security, though he retains an interest in multimedia with recent work in networked animation systems. He has

consulted in the area of security and is currently involved in projects exploring the crossover between AI and security in robust systems, and secure ambient IPv6 systems, though he has further interests in human factors as applied to the design and implementation of secure systems. Steve coordinates the activities of the mobile systems group at UCL.