JAMMY: a Distributed and Dynamic Solution to Selective Jamming Attack in TDMA WSNs

Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, Sajal K. Das

Abstract—Time Division Multiple Access (TDMA) is often used in Wireless Sensor Networks (WSNs), especially for critical applications, as it provides high energy efficiency, guaranteed bandwidth, bounded and predictable latency, and absence of collisions. However, TDMA is vulnerable to selective jamming attacks. In TDMA transmission, slots are typically pre-allocated to sensor nodes, and each slot is used by the same node for a number of consecutive superframes. Hence, an adversary could thwart a victim node's communication by simply jamming its slot(s). Such attack turns out to be effective, energy efficient, and extremely difficult to detect. In this paper, we present JAMMY, a distributed and dynamic solution to selective jamming in TDMA-based WSNs. Unlike traditional approaches, JAMMY changes the slot utilization pattern at every superframe, thus making it unpredictable to the adversary. JAMMY is decentralized, as sensor nodes determine the next slot utilization pattern in a distributed and autonomous way. Results from performance analysis of the proposed solution show that JAMMY introduces negligible overhead yet allows multiple nodes to join the network, in a limited number of superframes.

Index Terms—WSNs, TDMA, Security, Selective Jamming, DoS, Secure Slot Permutation, Decentralized Slot Acquisition

1 Introduction

Wireless Sensor Networks (WSNs) are being used in a variety of domains including industrial applications, factory automation, environmental and health monitoring and critical infrastructures. In such applications, Time Division Multiple Access (TDMA) is often used to access the shared wireless medium. In TDMA, time is divided into a sequence of periodic *superframes*, each consisting of a fixed number of transmission *slots*. Typically, slots are allocated to sensor nodes such that each node needs to be active only during its own slot(s), while it can sleep for the rest of the time. It is known that TDMA provides guaranteed bandwidth, high energy efficiency, absence of collisions (i.e., reliability), as well as bounded and predictable latency.

Unfortunately, TDMA suffers from *selective jamming* attack, a particularly insidious form of Denial-of-Service (DoS) that allows an adversary to completely thwart the communication of a victim node with a very low probability to be detected. In TDMA-based WSNs, a node typically retains its slot for many consecutive superframes. Therefore, an adversary could preliminarily monitor communication and detect the slot of the victim node. Then, the adversary could jam that slot – feigning a collision, for example – and sleep for the rest of the superframe. Such an attack is very effective, energy efficient, and much more difficult to be detected than a traditional wide-band jamming [33]. Although several methods are available for jamming detection [36], their applications to selective jamming is greatly complicated by the limited exposure time of the adversary and the limited amount of traffic affected by the attack.

Both physical-layer solutions and cyber countermeasures against selective jamming have been proposed in the literature, but they all exhibit some limitations. For example, solutions operating at the physical layer rely on spread-spectrum techniques, increased transmission power, and antenna polarization or directional transmission [3][7]. Unfortunately, they only make the attack more difficult, but are not able to neutralize it. Furthermore, these solutions are more suitable for military networks with a large design space, whereas commercial networks do not have the same flexibility, since they must conform to norms and laws. Cyber countermeasures for WSNs include [1][4][25]. However, they are mainly tailored to the IEEE 802.15.4 standard and, hence, are not general. Furthermore, DEEJAM [1] is based on frequency hopping, redundant encoding, and packet fragmentation, and thus introduces a significant computing and communication overhead. Finally, [25] is a totally centralized solution, where sensor nodes regularly exchange messages with a Coordinator node, and hence displays a considerable energy consumption.

To overcome these limitations, we propose JAMMY, a novel distributed and dynamic solution against selective jamming attacks in TDMA-based WSNs. The proposed methodology is based on a key idea, i.e., randomly permuting the slot utilization pattern on a superframe basis. By doing so, the slot(s) used by a sensor node change(s) *unpredictably* at each superframe. Hence, the adversary is forced to jam slots picked at random in the hope to

- *M. Tiloca is with SICS Swedish ICT AB, Kista, Sweden (email: marco@sics.se).*
- *D. De Guglielmo, G. Anastasi and G. Dini are with the Dept. of Information Engineering, University of Pisa, Italy (email: {d.deguglielmo, g.anastasi, g.dini}@iet.unipi.it).*
- *S. K. Das is with the Dept. of Computer Science, Missouri University of Science and Technology, Rolla, MO, United States (e-mail: sdas@mst.edu).*

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The research leading to these results has received funding from the European Union *Seventh Framework Programme (*FP7/2007-2013*) under* grant agreement *n° 246016.*

guess the ones used by the victim node. Assuming that a single slot per sensor node is used at each superframe, the probability of a successful selective jamming attack is $1/N$, where N is the number of slots in a superframe. JAMMY is *distributed*, as each sensor node computes the slot to use in the next superframe autonomously (i.e., without exchanging data) and in a *consistent* way (i.e., without causing collisions). JAMMY is also *dynamic* as it manages dynamic join and leave of multiple nodes. Finally, JAMMY is *general* as, in principle, it can be used in any TDMA network.

We evaluated the performance of our proposed solution through both simulation experiments and measurements in a real large-scale testbed. We also compared it with a centralized solution, both in steady-state and dynamic conditions. Our results show that JAMMY is effective. It is also *efficient* as it introduces a negligible processing overhead and no communication. Also, JAMMY allows new sensor nodes to join the network in a limited time, with consequent benefits in terms of energy consumption.

The paper is organized as follows. Section 2 surveys the related works. Section 3 introduces the system model and the attack model. Sections 4, 5 and 6 present JAMMY. Specifically, Section 4 describes how sensor nodes compute the next slot utilization pattern in a distributed way, while Sections 5 and 6 describe how JAMMY manages the leaving and joining of sensor nodes, respectively. Section 7 analyzes the performance of JAMMY in steady-state conditions. Section 8 analyzes its performance in dynamic conditions. The results in dynamic conditions are presented in Section 9. Finally, conclusions are drawn in Section 10.

2 Related work

Jamming is considered one of the most common DoS attacks, as well as a severe security issue in wireless communications [14][20][30][33][34][36][37]. In the context of WSNs, jamming attacks aim at interfering with the network's operational frequencies. Xu *et al.* [36] have classified jamming attacks in WSNs as *constant*, *deceptive*, *random*, and *reactive*.

The objective of a *constant* jammer is to corrupt *all* network packets, by continually transmitting random signals. However, such an "always-on" jamming strategy is based on the continuous presence of a high interference level, hence it is easy to detect [34][36]. On the other hand, a *deceptive* jammer injects a constant stream of bytes into the network, making it look as legitimate traffic. Unlike constant jamming, deceptive jamming is harder to detect using monitoring tools, since legitimate traffic is sent on the medium. The main disadvantage of both the aforementioned jamming strategies is their power inefficiency that limits the attacker's ability to be perpetual (i.e., without depending on an external power source). In this regard, a more efficient strategy is *random* jamming. It consists of alternating sleep and jamming phases, thus reducing power consumption, but this is usually less effective than constant and deceptive jamming. Finally, a smarter and more power efficient approach is *reactive* jamming, which performs jamming only when transmissions from other nodes take place. Reactive jamming is likely to be confused with regular collisions, hence it is much more difficult to detect.

In [4][25][28], the authors considered *selective jamming*, a particular form of reactive jamming aimed at disturbing communication among sensor nodes according to specific criteria and objectives. Among the above jamming strategies, selective jamming is more difficult to detect due to the reduced adversary exposure, as well as more power efficient. In this paper, we focus on a specific type of selective jamming, where the adversary aims at disrupting communication of one particular sensor node. Such an attack is very easy to perform in a TDMA-based WSN. Since a node typically retains a slot for many consecutive superframes, an adversary has to monitor communication, detect such a slot and jam it in order to completely thwart the node's communications. Also, the attack is power efficient, as the adversary has to activate her radio only during slots used by the victim node.

Approaches against jamming proposed in the literature can be divided into *physical-layer* solutions and *cyber* countermeasures (our proposed solution JAMMY belongs to the latter class). Physical-layer solutions try to prevent a jammer from interfering with network operational frequencies. The most relevant proposals in this class have been surveyed by Mpitziopoulos *et al.* [3]. The authors mainly consider *Frequency-Hopping Spread Spectrum* (*FHSS*) [27], a spread-spectrum transmission method that switches a carrier among many frequency channels, according to an algorithm shared by the transmitter and receiver. Frequency hopping is based on the premise that operating on a channel orthogonal to the jammer suppresses the jamming interference. However, since current commercial systems use only a small number of orthogonal bands, and adjacent orthogonal channel interference exists, frequency hopping has been shown to be rather ineffective [18]. Additionally, *Direct Sequence Spread Spectrum* (*DSSS*) is also considered. It consists of multiplying data to be transmitted (RF carrier) by a Pseudo-Noise (PN) digital signal having a frequency (chip rate) much higher than the original signal. This replaces the original RF signal with a wide bandwidth signal displaying a spectrum equivalent to a noise signal, thus minimizing unauthorized interception and jamming of radio transmission between nodes. However, Mpitziopoulos *et al.* stress that, although the IEEE 802.15.4 standard [15] relies on DSSS, this does not make it invulnerable to jamming attack. Morever, the network is likely to be taken down by jamming, due to the limited supported chip rate, and the restricted transmission power of sensor nodes. The main drawback of physical-layer solutions is that they are not actually able to *neutralize* jamming attack.

Cyber countermeasures, on the other hand, assume that it is always possible for a jammer to interfere with network's regular transmissions and make use of security schemes to contrast jamming. The majority of solutions following this approach addresses constant jamming [7][21][34][35], while relatively fewer solutions target selective jamming [1][4][11][25]. In the following, we survey only cyber countermeasures addressing selective jamming. Our proposal JAMMY also falls in this category.

Wood *et al.* [1] proposed DEEJAM, a new Medium Access Control (MAC) protocol that provides defense against jammers using IEEE 802.15.4-based hardware. DEEJAM relies on frequency hopping, redundant encoding and packet fragmentation, and aims at hiding packets from a jammer node, thus evading its search and limiting the impact of packets that are corrupted anyway. DEEJAM is compatible with existing nodes' hardware. However, it is a solution specifically tailored to 802.15.4 WSNs and introduces significant computational and energy costs in resource constrained sensor nodes.

Proano *et al.* [4] analyze a specific selective jamming attack, where the adversary thwarts the transmission of particularly important kinds of packets. They also proposed some methods, based on cryptographic primitives, to mitigate the attack effects. Encryption of transmitted packets is an effective solution against packet classification. However, it requires that the entire packet, including the header, is encrypted (it is a common practice to leave the header unencrypted, so that receivers can early abort the reception of packets not destined to them). In their work, Proano *et al.* considered a jammer that continuously senses and classifies packets, in order to perform selective jamming based on their importance. Instead, in this paper we consider a different model of jammer, where the attacker does *not* need to continuously monitor the channel to effectively perform jamming attack.

In [11], Ashraf *et al.* proposed *Jam-Buster*, a low overhead framework against selective jamming. Jam-Buster relies on multi-block payloads, equal size of packets, and randomization of nodes' wake up time, to eliminate differentiation of packet types and reduce predictability of transmission times. Hence, the adversary is forced to transmit more jamming signals, and thus spend more energy to be effective. Also, more jamming transmissions eventually result in a faster detection of the jamming source. Jam-Buster does not try to outsmart the adversary through an actual anti-jamming solution, but focuses on making selective jamming less efficient and convenient to perform.

The closest work to JAMMY is [25], which proposes a countermeasure against the GTS-based selective jamming in 802.15.4 networks. GTS is basically a form of TDMA communication, where up to 7 *reserved* time slots in each superframe are allocated to sensor nodes by a central *Coordinator*. GTS is extremely vulnerable and prone to selective jamming attack [15]. In [25], the authors proposed a centralized solution where the slot utilization pattern is computed, and randomly changed at each superframe, by the Coordinator node. This reduces the attack effectiveness to at most $1/7$. However, the central Coordinator represents a single *point of failure*. In addition, this solution is tailored to IEEE 802.15.4 and, hence, not general. In contrast, JAMMY is distributed, because it does *not* require the presence of any central entity, although multiple nodes are still able to leave and join the network at any time. Finally, although JAMMY is proposed in the context of WSNs, it is a general solution that, in principle, can be used in any TDMA-based wireless network.

A preliminary, non optimized, version of JAMMY is presented in [22], which focused on single-hop WSNs; new sensor nodes were allowed to join only one at a time, and those already present in the network were required to transmit additional information at every superframe. On the other hand, the enhanced solution presented in this paper considers multi-hop WSNs and allows multiple nodes to simultaneously join at any time. Also, it does not require to transmit any additional information to counteract selective jamming, thereby introducing no communication overhead.

3 System and adversary model

This section introduces the system and adversary models, and describe the selective jamming attack considered in the paper. We consider a multi-hop WSN represented by a communication graph $G = (U, L)$, where $U = \{u_1, \ldots, u_n\}$ is the set of nodes in the network and $L = \{l_1, \ldots, l_m\}$ is the set of directed edges $l = (u_i, u_j)$, representing a *link* between node u_i and u_j. Specifically, an edge $l = (u_i,\ u_j)$ exists iff node u_i transmits data to node u_j.

In the considered WSN, nodes access the wireless medium using a TDMA method. This means that time is divided into periodic *superframes* of equal duration, each one of which is in turn composed of N equally-sized *slots*. In the following, s_i for $i = 1, \ldots, N$, will denote the i-th slot in the superframe. Slots are used by sensor nodes for transmitting/receiving data packets. Specifically, each sensor node remains active only during its own slots while it sleeps in the remaining time. Each slot is long enough to allow the transmission of a maximum-size data packet and the reception of the related acknowledgement (ACK). For simplicity, and without loss of generality, we assume that, in any superframe, a sensor node can use at most *one* slot to transmit data (*Uniqueness Property*) and uses it to communicate with a *single* intended receiver.

Many links can be simultaneously active during the same slot provided that they do not interfere with each other. In particular, at every link, no collisions have to occur during both the data packet and ACK transmission (*Collision-Free Property*). In other words, for every link $l = (u_i, u_j) \in L$, it must be guaranteed that, when link (u_i, u_j) is active, i) no other node within the interference range of u_j transmits data; and ii) no other node within the interference range of u_i receives data (and, hence, sends ACKs). For each link $l \in L$, we define a set of interfering links $\mathcal{I}(l)$ which includes all the links belonging to L that interfere with l (note that $\mathcal{I}(l)$ contains l itself). Thus, the Collision-Free condition is straightforwardly defined as follows:

$$\forall \text{ slot } s, \sum_{i \in \mathcal{I}(l)} x_i(s) = 1 \text{ if } l \text{ is active during } s \tag{1}$$

where $x_l(s)$ is a binary variable, such that $x_l(s) = 1$ if link $l \in L$ is active during s, and 0 otherwise. This means that, if link l is active during slot s, the associated interfering set $\mathcal{I}(l)$ contains one active link only, i.e. l itself.

We also assume that the considered WSN has a dynamic membership, i.e. sensor nodes may join and leave dynamically. Specifically, a sensor node joins the WSN when its mission starts, and leaves it when its mission terminates. Upon joining the WSN, a node runs a *decentralized Slot Acquisition algorithm* in order to acquire a slot to be used for its subsequent data transmissions. We do not commit to any specific slot acquisition algorithm, provided that the adopted one satisfies the Uniqueness and Collision-Free properties as defined above. We further assume that a node retains the acquired slot until it leaves the network. Upon leaving, it releases the slot, which becomes available to other joining nodes.

With reference to the above system model, we consider an *external* adversary whose objective is to disrupt all transmissions of *one* specific victim node, by performing a *selective jamming* attack, i.e., by maliciously transmitting during the victim's transmission slot. We assume that the adversary does not compromise any sensor node, either physically or logically, but she is able to eavesdrop and jam any communication within the WSN. In addition, while performing the attack, the adversary is willing to be as much invisible as possible, in order to limit the likelihood of being detected, and to save as much energy as possible. Specifically, the considered adversary can easily succeed in playing the selective jamming attack under the aforementioned constraints as follows. First, she monitors communications for one or more superframes, and identifies the slot used by the victim node for data transmission. The specific approach that the adversary adopts to perform this task is not important here. For instance, she may exploit some prior knowledge such as the victim's identifier, its position in the network, or the type of traffic it produces. Then, starting from the next superframe, the adversary systematically jams that slot as follows. The adversary stays quiet until the victim's slot and starts transmitting a radio signal as soon as it senses the victim's activity on the channel. This behavior features a form of reactive jamming that is harder to detect [33]. Since the victim uses the same slot in all superframes, the attack is 100% *effective*. Besides, it is also *energy-efficient*, as the adversary has to jam only one slot per superframe, while she can turn off her radio during all other slots. Finally, the attack becomes hardly-detectable, as it exposes the adversary for a very limited amount of time (one slot per superframe).

Throughout the paper, we refer to *jammed area* as the portion of the WSN within which no packets can be correctly received during the attack performance. Note that, in order to successfully carry out the selective jamming attack, the jammed area must include the receiver node associated with the victim node.

4 The JAMMY algorithm

The selective jamming attack described above is based on the observation that, in a traditional TDMA approach, the same slot is typically used by a sensor node for data transmission for many consecutive superframes (e.g., until the node leaves the network). Hence, one way to contrast the attack is to change the slot utilization pattern at every superframe, so making it *unpredictable*. This means that the adversary must not be able to predict the slot to be used by the sensor node in the next superframe, even after observing a number of superframes. Thus, the only strategy available to the adversary to perform the selective jamming attack while retaining power-efficiency and hard-detectability is to randomly pick a slot and jam it. It follows that the attack effectiveness decreases to $1/N$, where N is the superframe size (see Section 7).

In order to achieve such a goal, at the end of every superframe we can compute the next slot utilization pattern as a *random permutation* of the current one. However, *unpredictability* is not sufficient. We also require that sensor nodes are able to compute the next utilization pattern *autonomously*, i.e., relying only on locally available information. In addition, the computation of the slot utilization pattern must be *consistent*. That is, all the nodes in the network must autonomously compute the *same* permutation. Otherwise, collisions would occur and the Collision-Free property would not be guaranteed anymore.

To fulfill such requirements, we assume that every node executes a *random permutation algorithm*. At the end of each superframe, every node randomly permutes the current slot utilization pattern, thus producing the next one. Typically, a random permutation algorithm uses a random number generator. In order to prevent collisions, nodes must compute the same permutation and, thus, have to produce the *same* sequence of random numbers. It follows that nodes must use *pseudo-random number generators* which must be maintained in the same internal state. This also implies that, when a new node joins the network, its generator must be initialized into the same internal state as the one of the nodes already in the network. Also, to fulfill the unpredictability requirement, the sequence of psedo-random numbers must also be unpredictable, and therefore the pseudo-random number generator must be secure [2].

In the following, we present our countermeasure against selective jamming in detail. Specifically, in Section 4.1, we introduce the random permutation algorithm and the secure pseudo-random number generator (SPRNG) used in JAMMY. Then, we analyse the system in steady state condition, i.e., when no sensor nodes join/leave the network (Section 4.2). Finally, we relax this assumption and show that sensor nodes may join and leave at any time without jeopardizing the solution (Sections 5 and 6).

It is worth noting that an adversary could still completely jam the network by performing a wide-band jamming. Alternatively, she could continuously monitor the network in order to detect the new slot used by the victim node, and then jam it. However, by doing so, she would compromise her hard-detectability and power efficiency. Specifically, a wide-band jamming would make the adversary considerably easier to be detected [33]. Furthermore, wide-band jamming and continuous monitoring would increase the adversary's power consumption, thus making the attack inconvenient from the energy point of view.

4.1 On Implementing a Random Permutation

In this section, we introduce the two basic components of JAMMY, namely a random permutation algorithm and a Secure Pseudo-Random Number Generator (SPRNG). While the literature provides many instances of such algorithms, we need to design two of them which are affordable on resource constrained sensor nodes. As to the random permutation algorithm, we have used the Fisher-Yates algorithm [19], also known as the Knuth shuffle algorithm , which runs in $\mathcal{O}(n)$ time. We have implemented the SPRNG by means of a block cipher in the counter mode (Algorithm 1) [2]. Let $E(x, y)$ denote a cipher which encrypts a plaintext y by means of a key x. First, we provide the generator with an encryption key K, and initialize a counter z to a random seed z_0. Then, we apply the cipher to the sequence of values $z, (z+1), (z+2), \ldots$, so producing the output random sequence $E(K, z), E(K, z+1), E(K, z+2), \ldots$. In the following, we call counter z the *internal state* of the generator, and K the *permutation key*. We assume also that K is kept secret and its length discourages an exhaustive key search.

This is a common method to build a SPRNG out of a cipher [2][5]. The crucial design requirement is that the cipher must be secure. Here we refer to the AES, which has the following two advantages. The first one is security: there is currently no known analytical attack against AES with a complexity less than a brute-force attack [5]. The other advantage is that AES is affordable on resource-constrained sensor nodes. Besides, commercially-available sensor node platforms such as Tmote Sky provide AES-128 encryption in hardware, with negligible overhead in terms of delay, storage, and energy consumption [24]. An optimized random permutation algorithm, for transmitter-only nodes, is reported in the Appendix A.

```
unsigned  K; // permutation key
unsigned  z; // counter
unsigned  random()
{
  unsigned val = E(K, z);
  z = (z + 1);
  return val;
}
```

Algorithm 1: Secure Pseudo-Random Number Generator.

4.2 The Secure Slot Permutation Algorithm

We are now in a position to describe the Secure Slot Permutation (SSP) algorithm used by JAMMY to protect communications against selective jamming attack. In this section, we assume that, after the system initialization, the WSN membership is static, i.e., no sensor nodes join/leave.

We assume that every sensor node maintains a *permutation vector*, namely a vector of N unsigned elements, which represents the node's view of the current slot utilization pattern. We denote by v_u the permutation vector of node u where $v_u[i]$ refers to the i-th slot in the superframe. Every node maintains its own permutation vector as follows. If node u does not use slot s_i, then $v_u[i] = 0$. If node u uses slot s_i to transmit data, then $v_u[i] = 1$. Finally, if node u uses slot s_i to receive data from an associated transmitter, then $v_u[i] = 2$. The Uniqueness Property implies that the permutation vector v_u of a node u contains just one element that is equal to 1. More formally, $\forall\ 0 \leq i, j < N, v_u[i] = v_u[j] = 1 \Leftrightarrow i = j$. Finally, the Collision-Free Property implies that all links active in a given slot s_i do not interfere with each other. More formally, for any pair of links $l_1 = (a, b),\ l_2 = (c, d)$ active during slot s_i, i.e., $v_a[i] = v_c[i] = 1$ and $v_b[i] = v_d[i] = 2$, we have $l_1 \notin \mathcal{I}(l_2)$ and $l_2 \notin \mathcal{I}(l_1)$, that is l_1 and l_2 do not interfere with one another. Notice that element i can be 0 in every permutation vector iff slot s_i is not associated to any node.

Let us assume that nodes have been initialized via off-line methods so that they all secretly share the same permutation key K, and their on-board SPRNGs are all initialized to the same initial state z_0. Quantity K and z_0 are randomly selected following the recommendations in [16]. Finally, an initial slot utilization pattern satisfying the Uniqueness and Collision-Free properties has been defined and permutation vectors have been initialized accordingly. Without any loss of generality, we may assume that this initial slot utilization pattern has been defined off-line. Alternatively, it may have been produced by the decentralized Slot Acquisition algorithm described in Section 6.1.

```
   Upon current superframe expiration:
1: randomly permute $v_u$
2: Build set $T$ s.t. $T = \{i :\ v[i] = 1\}$
3: Build set $R$ s.t. $R = \{j :\ v[j] = 2\}$
4: return $T$, $R$
```

Algorithm 2: Secure Slot Permutation.

Since initialization, each node protects itself from the selective jamming attack by *periodically* performing the *Secure Slot Permutation* (SSP) algorithm (Algorithm 2) at the end of every superframe. The SSP algorithm takes a permutation vector as input and (pseudo-)randomly permutes it (line 1). Then, it builds two sets, T and R (lines 2 and 3). Specifically, either T is an empty set (if u is a receiver-only node) or it contains the index of the slot to be used for transmission in the next superframe (Uniqueness property). Instead, R contains the indexes of the slots to be used to receive data during the next superfame. Then, Algorithm 2 returns sets T and R (line 4).

To fix ideas, let us focus on the first execution of the SSP algorithm, i.e., at the end of the first superframe. When this superframe ends, every node executes the SSP algorithm passing its permutation vector as input. As all nodes share the same permutation key K and have the SPRNG in the same state (z_0), they compute the *same* permutation, thus meeting the requirement of consistency. Since the permutation is based on a SPRNG, then it results unpredictable for an adversary who does not know the permutation key, i.e., the adversary cannot predict the slot used by the victim node to transmit data in the next superframe. Note that the only way for the adversary to compute the permutation is to obtain the permutation key. However, this is not possible since the adversary has no access, neither logically nor physically, to the sensor node's memory and the key has a size that discourages any brute force attack. The SSP algorithm operates only on locally available data and, thus, each sensor node can autonomously compute the permutation without exchanging data with other nodes. Also, every execution of the SSP Algorithm causes the counter of the SPRNG to be incremented by N. As all nodes compute the permutation at the end of the superframe, at the end of the first superframe, all SPRNGs are still in the same state, namely $z = z_0 + N$. It follows that, in the next execution of the SSP algorithm (end of the second superframe) nodes will compute the same permutation once again, and take their generators into the same next internal state. This reasoning can be iterated for any subsequent superframe, i.e., after r superframes, the internal state of the SPRNG will be $z = z_0 + r \cdot N$. As it turns out, the value of the counter of a SPRNG grows at a speed that is equal to the number of slots N in a superframe.

Note that, the size of the counter of the SPRNG establishes an upper bound to the maximum length of the random output sequence the generator is able to produce. Therefore, the counter size must be adequately large to avoid the counter to wrap-around during the network lifetime (e.g., 64–128 bits). Anyhow, one way to deal with the counter wrap-around is to refresh the permutation key and re-initialize generators. Since the internal states of all SPRNGs remain synchronized over time, the counter wrap-around occurs at the same superframe on all sensor nodes. Hence, at that point in time, all sensor nodes can simultaneously and autonomously generate a new permutation key K^+ as $K^+ = E(K, K)$. Thereafter, all sensor nodes rely on K^+ until the next counter wrap-around occurs.

Finally, we argue that the Uniqueness and Collision-Free properties are maintained.

Claim 1. *The SSP algorithm maintains the Uniqueness and Collision-Free properties at every superframe.*

Proof: Omitted. See Appendix B. □

5 Node leave

In this section, we describe how JAMMY behaves when sensor nodes leave the network. Upon leaving, a given node u stops using all slots it acquired for data transmission/reception, and does not perform any further action. The behavior of remaining nodes which were not communicating with u is not affected at all. Conversely, every node t involved in data communication with u behaves as follows. Let T_l denote the last superframe during which node u was active. Also, let us indicate as T_{l+k} the superframe when t realizes that node u has left the network. Finally, we refer to s_u as the slot that node t is supposed to use to communicate with node u during superframe T_{l+k}. Since u is not active anymore, t updates its own permutation vector as $v_t[u] = 0$, hence slot s_u becomes idle.

There are different ways for node t to realize that node u has left the network. For instance, node t can assume that node u is not active anymore if no successful communication with u occurs for k consecutive superframes. Alternatively, node u can explicitly alert node t about its own leaving, by means of a dedicated flag in its last data/ack packet transmitted to node t.

In order to assure and maintain network security, it is necessary to provide a new permutation key K to the remaining sensor nodes, by excluding and, thus, logically evicting the leaving ones. This can be done by rekeying, i.e., by revoking the current permutation key and distributing a new one to all nodes but the leaving one. Rekeying is beyond the scope of this paper. The literature provides many rekeying schemes for WSNs, including [12][13][29]. JAMMY does not pose any particular requirement on rekeying, thus we assume that it employs any available one.

6 Node join

JAMMY allows sensor nodes to join the network at *any* time. In addition, *multiple* nodes can join the network at the same time by executing a specific *join procedure* (Section 6.2). The *join procedure* assumes that every joining node performs a *Slot Acquisition* algorithm to get a slot in the superframe for data transmission. In principle, any decentralized slot scheduling algorithm could be used for this purpose. In this paper, we propose the following Slot Acquisition algorithm.

6.1 Slot Acquisition algorithm

We assume that joining nodes use a distributed approach, through which they autonomously acquire a slot in the superframe to communicate with a specific intended receiver. The algorithm considered here is an extension of the one presented in [6] which was conceived for single-hop networks. In the following, we refer to a joining node u which wants to acquire a slot to communicate with a receiver node t. For simplicity, we assume that node t is active (i.e. it keeps its radio on) during all slots in the superframe. Since we assume that an arbitrary number of sensor nodes attempts to join the network simultaneously, some of them will compete with node u to acquire a slot for transmitting data to node t, while others will refer to different receivers. However, if two joining nodes are physically close to each other, they might interfere during the slot acquisition phase, as described below.

According to our algorithm, sensor nodes compete to acquire one slot to transmit data to their respective destination node. Specifically, the joining node u contending to acquire a slot s_i tries to transmit a *fake* packet to node t during this slot. If t successfully receives such a packet, it sends back an ACK to u during the same slot s_i. If the ACK is successfully received by node u, the latter acquires the right to use slot s_i to transmit data to node t in all subsequent superframes. In fact, if both the fake packet and ACK transmissions are successful, then no interference with other links has occurred. Thus, u stops the slot acquisition process, and completes the join procedure (see Section 6.2). Starting from the next superframe, link (u, t) can use s_i.

Now, let us describe the case when node t has not successfully received any packet during slot s_i. This happens if no node has tried to transmit data to t during slot s_i, or if a collision has occurred. We have to distinguish two different cases. If $v_t[i] = 0$, i.e., slot s_i is idle, then node t performs no further actions. Otherwise, if $v_t[i] = 2$, i.e., slot s_i has been already acquired by a transmitter node from which t expects to receive a data packet, then node t broadcasts an *Alert* packet. This is to notify all joining nodes within the interference range of t that slot s_i is not available.

The Slot Acquisition algorithm relies on a *random backoff time* to prevent collisions from competing joining nodes accessing the same slot s. Specifically, each joining node waits for a random backoff w in the range $\{0,\ 1,\ \ldots, W_B - 1\} \cdot D_{bo}$, where W_B is the backoff window size and D_{bo} is the backoff unit and, then, checks the channel status. If the channel is found idle, it transmits the fake packet. Thus, four possible outcomes can occur: (**i**) *ACK_RECEIVED*; (**ii**) *CHANNEL_BUSY*; (**iii**) *ALERT*; or (**iv**) *NO_NOTIFICATION*. In case of successful ACK reception (case (**i**)), node u acquires slot s to transmit data to node t, as described above. Note that, once nodes have acquired a slot, in all subsequent superframes they do not wait for any backoff time nor check the channel status before transmitting their data packets. This assures that active nodes have priority over joining nodes. If node u finds the channel busy after the backoff time (case (**ii**)), it means that other nodes in the proximity of u are already using slot s. Specifically, one or more joining nodes physically close to u have selected a *shorter* backoff time (i.e., node u has lost the contention), or slot s has been already acquired by a transmitter node. Similarly, if u receives an *Alert* packet (case (**iii**)), it means that slot s is not available, as it is used by another transmitter node which is not in the sensing range of u. In both the cases (**ii**) and (**iii**), node u tries to acquire the next slot $s{+}1$ in the current superframe. The last case is when no notification is received by node u (case (**iv**)). This happens when: **a)** a link interfering with (u, t) is using slot s; or **b)** slot s is available but the fake packet of u collided with the fake packet sent by another joining node. In such a case, node u tries again with the same slot s in the next superframe. The rationale behind this is that, if a collision occurred and the number of colliding sensor nodes is limited, the contention will very likely be resolved during the next superframe.

```
1: Choose a slot $s$ in $[1, N]$ randomly;
2: Contend for $s$ (using a random backoff $w$);
3: Case ACK_RECEIVED:
4:     Acquire $s$ and terminate the Slot Acquisition process;
5: Case CHANNEL_BUSY or ALERT:
6:     Re-try $s{+}1$ (with new random backoff $w$);
7: Case NO_NOTIFICATION:
8:     Defer contention to $s$ in the next superframe (with new random backoff $w$);
```
Algorithm 3: Slot Acquisition.

Algorithm 3 shows the actions performed by a joining node u. Initially, it selects a random slot s in the current superframe, in order to perform the contention process. This random choice is aimed at spreading contention attempts within the whole superframe, thus reducing the number of competitors for every single slot, and increasing the success probability. Then, node u contends for slot s using a random backoff time w. As it can be observed, the slot acquisition process can take more than one superframe to complete. Also, note that the Secure Slot Permutation (Section 4.2) complicates the acquisition of a slot, since the slot utilization pattern changes at every superframe.

6.2 Join procedure

In this section, we describe the complete set of actions that any joining node u performs to correctly join the network, including the execution of the Slot Acquisition algorithm described in Section 6.1.

To correctly start the join process at superframe T_j, we assume that node u is provided with i) the shared permutation key K; and ii) the value z_j to initialize the generator counter z. Node u can retrieve such security material from an additional entity, namely *Join Manager*, which is responsible for the correct initialization of joining nodes. In principle, the Join Manager can be implemented in both centralized and distributed fashions. Intuitively, a distributed version of the Join Manager can be composed of a set of replicas, each one of which i) holds both the current permutation key and SPRNG state; ii) keeps itself synchronized with superframes in order to maintain an up-to-date value of the SPRNG state; and, iii) participates to rekeying in case of node's leaving. For the sake of space, here we consider a centralized version. One may argue that this may constitute a single point of failure. However, we can reasonably assume that the Join Manager is a special-purpose computer properly designed, implemented and managed to be reliable and secure. Although server reliability and security are still research issues, the literature provides a number of established techniques and methodologies (e.g. [10][17][26]). The centralized Join Manager keeps itself synchronized with superframes to maintain an up-to-date value of the SPRNG state. Besides, it participates to rekeying in case of node's leaving (see above).

```
1: $z \leftarrow z_j$
2: $v_u \leftarrow 0$
3: $s_i \leftarrow SlotAcquisition$
4:     □ handler upon(superframe expiration)
5:             $z \leftarrow z + N$
6: $v_u[i] \leftarrow 1$
```
Algorithm 4: Join procedure.

Algorithm 4 describes the specific actions performed by node u during the join procedure. Initially, u initializes its generator to z_j and its permutation vector to 0 (lines 1-2). Then, u executes the Slot Acquisition algorithm (Section 6.1) to acquire a slot s_i for transmitting data to its intended receiver t (line 3). The Slot Acquisition process may take one or more superframes to be completed. Then, once initialized, the generator counter has to be kept up-to-date with respect to the one on the other nodes. Therefore, while the slot acquisition process is in progress, we activate a handler (line 4) that updates the generator counter whenever a superframe expires (line 5). Finally, once the slot acquisition process has been completed, node u updates its permutation vector to reflect such a slot acquisition (line 6). Note that node t has to update its own permutation vector v_t. Specifically, upon successfully receiving the fake packet from node u during slot s_i, node t updates its own permutation vector v_t as $v_t[i] = 2$. This ensures that t considers slot s_i as reserved for communication with u. Hereafter, at the end of every superframe T_m, node u locally determines the slot to be used in the next superframe T_{m+1}, according to the *Secure Slot Permutation* algorithm described in Section 4.2.

7 Analysis in Steady State Conditions

In this section, we consider a WSN operating in steady state conditions (i.e., no sensor node joins or leaves the network), and investigate the effectiveness of our proposed solution against selective jamming. Furthermore, we compare JAMMY with a centralized solution, in terms of effectiveness and overhead introduced. For this purpose, we consider a centralized solution that generalizes the solution proposed in [25]. The centralized solution considered here relies on a *Coordinator* node that, at each superframe T_m: i) generates a random slot utilization pattern, namely S_m; and ii) broadcasts S_m together with a *Message Authentication Code*, thereby ensuring S_m authenticity and freshness. Then, every node retrieves S_m, and becomes aware of the specific slot it is supposed to access at T_m to transmit data. Since a new slot utilization pattern S_m is randomly created on a per-superframe basis, the only strategy available to the adversary is to randomly pick a slot and jam it.

7.1 Effectiveness Analysis

To evaluate the effectiveness against selective jamming, we consider the *attack success probability*, defined as the fraction of packets transmitted by the victim node u that are corrupted by the attacker. We consider both the case when no countermeasure is used, and the case where either the centralized solution introduced above or JAMMY is used. We recall that, in order to successfully carry out the selective jamming attack, the jammed area must include the receiver node associated with the victim node u. We assume that the communication channel is ideal. Hence, corrupted packets are only due to the selective jamming attack.

Our results are derived by means of simulation experiments. Each experiment consists of 10 independent replications, and, for each replication, $1,000,000$ superframes are considered. We averaged simulation results over all replications and derived confidence intervals by using the independent replication method and 95% confidence level.

Figure 1 shows the effectiveness of the two different approaches against selective jamming, i.e., the centralized solution and JAMMY, considering three different numbers of slots N in the superframe. We observe that, if no solution is used, all transmissions from node u, occurring during slot s_u, are corrupted by the jammer (i.e. attack success probability equal to 1). At the same time, the attack does not affect any other nodes $v \neq u$ which transmits in a slot $s_v \neq s_u$. Also, as expected, the attack success probability is independent of the number of slots in the superframe. Finally, the adversary also corrupts the transmissions from all other nodes $u' \neq u$ using the same victim slot s_u, and whose associated receiver is in the jammed area.

Conversely, when using a selective jamming countermeasure (i.e., the centralized solution or JAMMY), the adversary cannot track her target u anymore. Hence, the only available strategy consists of jamming one slot s_r, by picking it at random among the N slots in the superframe. As a consequence, the adversary corrupts a fraction $(1/N)$ of the transmissions from every node which uses the jammed slot and whose associated receiver node is in the jammed area. It follows that only $1/N$ of transmissions from the victim node u is corrupted by the jamming adversary, i.e., if $s_r = s_u$. Again, in such a case the adversary also corrupts the transmissions from all other nodes $u' \neq u$ which use the same victim slot s_u, and whose associated receiver node is in the jammed area. On the other hand, if $s_r \neq s_u$, the adversary ends up to jam transmissions from other nodes $v \neq u$ which use slot s_r and whose associated receiver node is in the jammed area. Hence, if a countermeasure is used, the considered selective jamming attack fails with probability $(N-1)/N$.

Let us consider the general case when $J > 1$ colluding adversaries cooperate to jam transmissions from node u. Specifically, J different slots are selected at random, and each slot is jammed by a different adversary. We assume that $J \ll N$, to fairly model the considered selective jamming attack, rather than a wide-band jamming. Hence, the J adversaries overall corrupt a fraction (J/N) of the transmissions from every node using one of the J jammed slots and whose associated receiver node is in the jammed area. If the J adversaries are not colluding (i.e. do not cooperate), it is possible that multiple adversaries jam the same slot. Thus, given the number of actually jammed slots $J^* \leq J$, the attack success probability is equal to (J^*/N).

Figure 2 reports the simulation results for the case of multiple jammers. Specifically, it shows the attack success probability for different numbers J of (colluding) jammers, when no solution is adopted and when either JAMMY or the centralized solution is used (the superframe size is equal to $N = 30$ slots). When no solution is adopted, all transmissions from the the victim node are corrupted by the jammers. Conversely, the results obtained when a selective jamming countermeasure is used are fully consistent with the expected theoretical value (J/N), i.e., 0.033, 0.10, and 0.167 for $J = 1$, $J = 3$, and $J = 5$, respectively.

Our results show that, from every node's perspective, the percentage of corrupted packets is acceptable and practically affordable, and thus can be suitably handled by means of retransmissions. Furthermore, the attack success probability decreases for greater values of N. This suggests that an appropriate sizing of the superframe would result in a practically ineffective selective jamming attack.

Finally, the centralized solution and JAMMY display the same effectiveness against the considered selective jamming attack. However, if the adversary interfered with

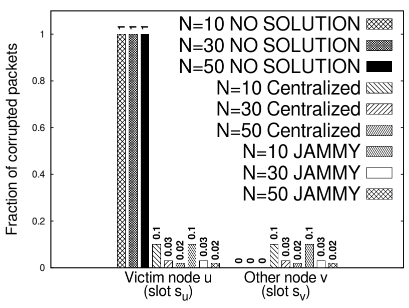

Fig. 1: Effectiveness against SJ ($J = 1$)

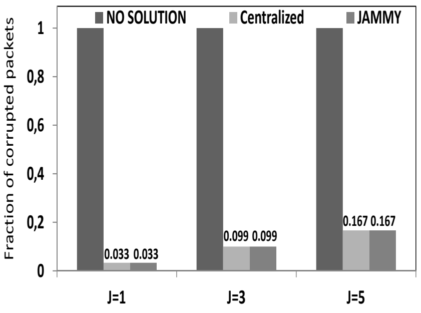

Fig. 2: Effectiveness againts SJ ($J \geq 1$)

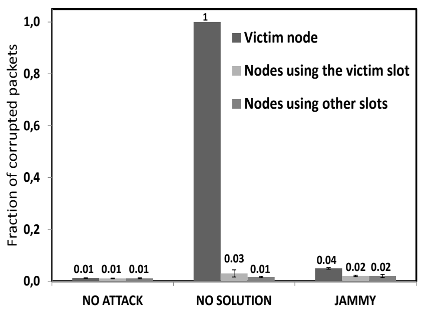

Fig. 3: Effectiveness against SJ (Indriya testbed)

the transmission of the slot utilization pattern S_m performed by the Coordinator node in the centralized solution, she would be able to prevent sensor nodes from receiving the current slot utilization pattern, thus compromising network communications altogether. Conversely, this cannot happen with JAMMY, as the latter is a distributed solution.

7.1.1 *Effectiveness analysis in a real WSN*

To validate our simulation results and evaluate the effectiveness of JAMMY also in a real WSN, we implemented our solution in the Contiki OS [9] and performed an experimental evaluation on the large-scale public testbed Indriya [8]. Indriya is composed of 97 TelosB sensor nodes deployed in a three-floor building at the National University of Singapore. For each experiment, we considered 100 different configurations, each consisting of a randomly-generated set of transmitter-receiver pairs of nodes in the testbed (i.e. links). To ensure a collision-free transmission schedule, we assigned TDMA slots to links offline, using an edge coloring algorithm [23]. For each network configuration, we also selected (randomly) the victim node and the jammer node. In all our experiments, we considered a single jammer and set the superframe size to $N = 30$ slots.

We investigated three different scenarios, namely NO ATTACK (there is no selective jamming attack), NO SOLUTION (the adversary performs a selective jamming attack and no countermeasure is used), and JAMMY (the adversary performs a selective jamming attack and JAMMY is used to contrast it). In order to experience similar conditions, we evaluated the three scenarios back to back. For each scenario, we ran 100 superframes with a certain network configuration and, then, we repeated the same steps for all the considered configurations. The results shown below are averaged over all the considered network configurations. We also show the 95% confidence intervals.

As above, to evaluate the attack success probability, we measured the fraction of corrupted packets experienced by nodes. However, in a real environment, the communication channel is not ideal and, hence, packets may be corrupted also due to communication unreliability. Hence, it is important to distinguish between packets corrupted by the jammer and packets altered by transmission errors. To this end, we considered for comparison the NO ATTACK scenario, where no attack occurs and packets are corrupted only due to transmission errors. Figure 3 shows that, in the considered experiments, the fraction of corrupted packets in the NO ATTACK scenario is very low, almost negligible.

Let us now focus on the two scenarios where a selective jamming attack occurs. Figure 3 shows that, when no countermeasure is used (NO SOLUTION scenario), all packets originated by the victim node are corrupted, as expected. In addition, transmissions from other nodes using the same victim's slot may be affected as well by the selective jamming attack, hence the fraction of corrupted packets slightly increases with respect to the corresponding value in the NO ATTACK scenario. Instead, there is no significant variation, with respect to the NO ATTACK scenario, in the fraction of corrupted packets experienced by nodes not using the victim's slot.

When using JAMMY to contrast the selective jamming attack, the fraction of corrupted packets experienced by the victim node (and, hence, the attack success probability) reduces to about 4%. Considering the small fraction of packets corrupted by transmission errors, this value is consistent with the expected theoretical value $1/N$ (corresponding to 3.3%, since $N = 30$ in our experiments). We also observe a reduction - with respect to the NO SOLUTION scenario - in the fraction of corrupted packets experienced by other nodes using the victim's slot. This is because, now, the adversary corrupts a significantly lower fraction of transmissions on that slot. Finally, nodes using a slot different from the victim's one experience a higher fraction of corrupted packets with respect to the NO SOLUTION scenario. This is because, with JAMMY, the transmissions from such nodes can be corrupted by the adversary, if they occur during the jammed slot. The results presented above confirm the ability of JAMMY to effectively contrast selective jamming attacks in a real WSN.

7.2 Overhead Analysis

We conclude our analysis in steady-state conditions by evaluating the overhead incurred by JAMMY, and comparing it with that of the centralized solution introduced above.

From a computational standpoint, JAMMY performs only simple encryption operations, during the execution of the SSP algorithm (see Section 4). Since such operations can be efficiently performed by common hardware platforms [32], the computing overhead introduced by JAMMY results to be negligible. Most importantly, from a communication standpoint, JAMMY does not require sensor nodes to perform any further transmission or reception (in addition to the initial reception of K and z_j from the *Join Manager*). This results in two main advantages. First, it does not determine any reduction of the available network bandwidth. Second, it does not affect the energy consumption of sensor nodes.

Let us now consider the centralized solution. For the purpose of our analysis, the *Coordinator* node represents a generic slot utilization pattern S_m as an array of U elements, where U is the number of active nodes in the WSN. In particular, the i-th element of S_m specifies the slot to be used at superframe T_m by sensor node i. Hence, each element of S_m has size equal to $\lceil log_2 N \rceil$ bits and the overall size of S_m is $U \cdot \lceil log_2 N \rceil$ bits. If we denote by C the size (in bits) of the Message Authentication Code, then the energy overhead introduced by the centralized solution at *each* superframe is the total energy E_{centr} spent by all sensor nodes to receive the slot utilization pattern. Thus,

$$E_{centr} = \frac{U \cdot P_{RX} \cdot ((U \cdot \lceil log_2 N \rceil) + C)}{R} \quad (2)$$

where P_{RX} is the radio power consumption in receive mode, and R denotes the data transmission rate. In the following, we refer to a data transmission rate $R = 250$ Kbit/s [15] and a power consumption $P_{RX} = 35.46$ mW [32].

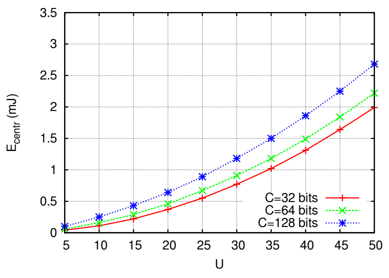

Fig. 4: Energy consumed by a centralized solution ($N = 30$).

Figure 4 shows E_{centr} for different numbers of active nodes (U) and different sizes of the Message Authentication Code (C). In particular, we refer to a number of slots per superframe $N = 30$. We can observe that the more nodes in the network, the more energy is consumed. Also, as expected, E_{centr} is higher for larger values of C.

Finally, note that E_{centr} is the constant amount of energy spent by the network at *each* superframe, hence the impact on energy consumption is constant over time. Instead, as discussed above, JAMMY does not result in any communication overhead other than the initial provisioning of K and z_j. Thus, its impact on the network lifetime is negligible, while resulting to be as effective against selective jamming as the centralized solution.

8 Analysis in Dynamic Conditions

In this section, we analyze JAMMY in dynamic conditions, i.e., when nodes join/leave the network. We develop an analytical model of the JAMMY join procedure, and use it to derive both the average energy and time spent by sensor nodes to join the network. For comparison, in Appendix E we also derive similar formulas for the centralized solution.

We consider a generic multi-hop WSN using JAMMY to counteract selective jamming, and focus on a scenario where a number N_j of sensor nodes starts the join procedure at the same superframe T_j, in order to acquire a slot in the superframe for communication (see Section 6.2). Let L_{max} represent the maximum distance between any two sensor nodes. We denote by R_{CS}, R_I and R_{TX} the carrier sensing, interference and transmission range of each sensor node, respectively. Since we are considering a multi-hop WSN, it follows that $R_{TX} < L_{max}$. Following the literature, we assume $R_{CS} \geq R_I > R_{TX}$. In addition, to simplify our analysis and make it tractable, we assume that $R_{CS} = R_I > L_{max}$. In such a case, all nodes in the WSN can sense and/or interfere each other's transmissions and, hence, the spatial reuse of slots is not possible. It follows that at most N links can be active in the WSN and, hence, this is a critical situation regarding slot acquisition.

At superframe T_j, we assume that N_A communication links are active in the WSN. It follows that $N_A \leq N$ slots of the superframe are already used while the remaining $N_F = (N - N_A)$ slots can be acquired by joining nodes for communication. In the analysis, we consider the case where $N_j = N_F$, i.e., the number of joining nodes is equal to the number of available slots. In Section 9.2, we also consider the case when $N_j > N_F$. We point out that, in general, JAMMY allows any number of sensor nodes to simultaneously join the network. Specifically, JAMMY guarantees that any joining node successfully terminates its join procedure in a short amount of time, as long as there is an available slot in the superframe to accommodate its transmissions. The analysis is divided into two parts. First, we consider all the events that can occur during the contention to acquire a slot. For each event, we derive the corresponding probability and the energy spent by contending sensor nodes. Then, we use such probabilities to derive a Discrete Time Markov Chain (DTMC) model of the overall join procedure and calculate the performance metrics of interest. We make the following assumptions about the Slot Acquisition algorithm (Algorithm 3).

1) A simple random backoff algorithm is used to solve contention among sensor nodes trying to acquire the same slot. Before sensing the channel and transmitting a fake packet, each sensor node waits for a random backoff time w in the range $\{0, 1, ..., W_B - 1\} \cdot D_{bo}$, where W_B is the backoff window size and D_{bo} is the backoff unit.
2) All joining nodes start competing at the first slot of superframe T_j, i.e., we do not consider the initial randomization (line 1). This maximizes the contention and, hence, models a worst case condition.

8.1 Event Probabilities

In this section, we consider all the events that can occur during the contention for a slot and, for each of them, we derive the corresponding probability and the energy spent by the contending sensor nodes.

Let us focus on a generic slot s^* and assume that M sensor nodes, out of N_j, are contending for s^*. First, let us consider the case when slot s^* is already used by an active link. Since active nodes have priority over joining nodes, all M joining nodes find slot s^* already busy. According to the Slot Acquisition algorithm (Algorithm 3), they wait for the next slot in the same superframe. Now, let us analyze the case when slot s^* is free (it is not currently used by any active link). Given the assumptions, the contention can result in one of the following outcomes.

(a) **SUCCESS**. One of the joining sensor nodes successfully transmits its fake packet during slot s^* and, hence, receives an acknowledgment.
(b) **COLLISION**. A collision is experienced by k joining nodes for $2 \le k \le M$, and no notification is received.
(c) **BUSY CHANNEL**. The channel is found busy by $h = (M-k)$ joining nodes, for $0 \le h \le (M-2)$, that schedule a retry at the next slot.

Please note that, since we are considering the case when $R_{CS} > L_{max}$, collisions between data packets and fake packets are not possible. It follows that the receiving nodes will never send $ALERT$ messages.

Now, we derive the probability of each of the above mentioned events to occur. Let W_B denote the backoff window size. In addition, we introduce the following definitions. We denote by: $P_s^F(M)$ the probability that a successful transmission occurs; $P_c^F(k \mid M)$ the probability that k out of the M contending nodes, $k \le M$, experience collision at the free slot; and, finally, $P_b^F(h \mid M)$ the probability that h joining nodes find the channel busy at the free slot. The following claim holds.

Claim 2.

$$P_s^F(M) = M \cdot \sum_{w=0}^{W_B-1} \left(\frac{1}{W_B}\right) \cdot \left(\frac{W_B-1-w}{W_B}\right)^{M-1} \quad (3)$$

$$P_c^F(k \mid M) = \binom{M}{k} \sum_{w=0}^{W_B-1} \left(\frac{1}{W_B}\right)^k \cdot \left(\frac{W_B-1-w}{W_B}\right)^{M-k} \quad (4)$$

$$P_b^F(h \mid M) = P_c^F(M-h \mid M) \quad (5)$$

Proof: Omitted. See Appendix C. □

Now, let us denote by P_{RX} (resp. P_{TX}) the power consumed by a sensor node in receive (resp. transmit) mode, and let D_{tx} and D_{ack} represent the duration of packet transmission time and ACK reception time, respectively. Also, let D_{CS} (D_{to}) denote the duration of the channel assessment (timeout interval). Finally, we indicate as $E_s(M)$ and $E_c(k \mid M)$ the total energy consumed when one of the M nodes transmits its packet successfully or k out of the M nodes experience a collision. Moreover, E_u is the energy spent by a sensor node during an acquired slot. The following claim holds.

Claim 3.

$$E_s(M) = M \cdot P_{RX} \cdot D_{CS} + P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{ack} \quad (6)$$

$$E_c(k \mid M) = M \cdot P_{RX} \cdot D_{CS} + k \cdot (P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{to}) \quad (7)$$

$$E_u = P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{ack} \quad (8)$$

Proof: Omitted. See Appendix C. □

8.2 Markov Chain Derivation

We are now in a position to derive a Discrete Time Markov Chain (DTMC) model of the JAMMY join procedure, that we use to derive the probability distribution of the joining time and the average energy consumed by sensor nodes.

We observe the system at the beginning of every superframe T_m, and represent the system state as a vector $X_m = [n_1,\ n_2, ...,\ n_N]$ where element n_i for $i = 1, \ldots, N$ refers to the i-th slot of the superframe T_m. Specifically, n_i indicates the number of joining nodes that contend for slot i during superframe T_m. We recall that we consider N_j sensor nodes joining the network at the same superframe T_j and assume that $N_j = (N - N_A)$, where N_A is the number of slots already acquired at superframe T_j. Since there are N_j joining nodes, we have $n_i \le N_j, \forall i$. Also, we denote by $S_m = [\overline{s}_1,\ \overline{s}_2,\ ...,\ \overline{s}_N]$ the slot utilization pattern at the beginning of a superframe T_m. Specifically, $\overline{s}_i$ indicates the state of slot s_i at superframe T_m, i.e., if it is free (F) and, hence, can be acquired by joining nodes or, if it is already acquired (A). Obviously, at superframe T_j, N_A slots must have state equal to A.

At the beginning of superframe T_j, the system state is $X_j = [N_j,\ 0, ...,\ 0]$, since all joining nodes contend for the first slot of superframe T_j (Assumption 2). Then, the system can evolve to different states depending on the slot utilization pattern at superframe T_j and the specific events that occur during superframe T_j. The join procedure terminates when the system reaches state $X_f = [0,\ 0, ...,\ 0]$, that is when all joining nodes have successfully acquired a slot for communication and, hence, become active nodes.

We designed an algorithm to **i)** derive all the possible states where the system can evolve to, **ii)** compute the probability that the system passes from one state to another, and **iii)** calculate the average energy spent by the system during each state transition. The algorithm takes as input N, N_A, N_j and produces as output set Ω and matrices P and E. Specifically, Ω is the set of possible system states, and P is the transition probability matrix of the system (i.e., each element P_{XY} where $\{X, Y\} \in \Omega$, indicates the probability that the system changes its state from X to Y). Finally, E refers to the energy consumption of joining nodes; each element E_{XY} is the average energy consumed by joining nodes when the system changes its state from X to Y. For the sake of space, we describe the designed algorithm in Appendix D. Here we use Ω, P, and E to derive:

- $P_{join}(k)$, $k = 0,\ 1,\ ...$: probability that the join procedure is over at the beginning of superframe T_{j+k}, i.e. the probability that *all* joining nodes complete their join procedure in k superframes. $P_{join}(k)$ provides the probability mass function of the joining time.
- $\overline{E_k}$, $k = 0,\ 1,\ ...$: average energy spent by *all* joining nodes during superframe T_{j+k}.
- $\overline{E_{join}}$: average energy consumed by *all* joining nodes during the entire duration of the join procedure. $\overline{E_{join}}$ represents the total join overhead in terms of energy.

To derive $P_{join}(k)$, we sort the states of the Markov Chain so that the initial state of the system $X_j = [N_j,\ 0, ...,\ 0]$ and the final state $X_f = [0,\ 0, ...,\ 0]$ are the first and last one in the sequence, respectively. Also, let v_0 be the *initial* probability vector, and $v_k,\ k \geq 0$, the probability vector related to superframe T_{j+k}. With no loss of generality, we can assume that $v_0 = [1, 0, \ldots, 0]$, thus $v_k = v_0 \cdot P^k$. Hence, the probability that the join procedure has been completed after k superframes, i.e., $P_{join}(k)$, corresponds to the probability that the system state at step k is X_f. Let us denote by $|\Omega|$ the cardinality of the set Ω, i.e., the total number of system states. Since X_f is the last state in the sequence, it follows that

$$P_{join}(k) = v_k[|\Omega|] \tag{9}$$

Let us now derive the average energy $\overline{E_k}$ spent by all joining nodes during superframe $T_{j+k},\ k \geq 0$. Let P_X^k be the probability that the system is in state X at superframe T_{j+k} , for any $X \in \Omega$. As P_X^k is the component of vector v_k associated to state X, then the following equation holds.

$$\overline{E_k} = \sum_{X \in \Omega} P_X^k \cdot \sum_{Y \in \Omega} E_{XY} P_{XY} \tag{10}$$

Equation 10 can be justified as follows. In order to calculate $\overline{E_k}$, we must consider all possible state changes that can occur from superframe T_{j+k} to the next superframe T_{j+k+1}. Hence, the outer sum considers any possible system state X, at superfame T_{j+k}, whose occurrence probability is P_X^k. Then, for each state X, the inner sum considers all possible states Y where the system can evolve to. Such a transition has a probability occurrence P_{XY}, and is associated with an average energy consumption E_{XY}.

Finally, we derive the average energy ($\overline{E_{join}}$) spent by all joining nodes during the whole join procedure. We recall that the join procedure is over when the system reaches state $X_f = [0,\ 0,\ ...,\ 0]$. Thus, $\overline{E_{join}} = \mu_{X_j}$ (11), where μ_{X_j} indicates the average energy consumed by all joining nodes to reach state X_f starting from state X_j. According to [31], the average energy μ_X consumed by the network to reach state X_f, starting from any state $X \in \Omega$, can be obtained by solving the following linear equation system, with μ_X as unknowns, and $\mu_{X_f} = 0$:

$$\mu_X = \sum_{Y \in \Omega} P_{XY} \cdot (E'_{XY} + \mu_Y),\ \forall X \in \Omega \tag{12}$$

where E'_{XY} differs from E_{XY}, since it does not take into account the energy consumed by joining nodes after they have completed the join procedure.

9 Performance Evaluation

In this section, we evaluate the overhead incurred by the join procedure, in terms of duration and energy consumption, by using the equations derived in Section 8. We also compare JAMMY with the centralized solution. To validate our analytical results, we rely on simulation experiments. Unless stated otherwise, the parameter values are as shown in Table 1. The considered values have been inspired by the 802.15.4 standard [15]. For simulation experiments, we used the same methodology described in Section 7 and considered 10 independent replications, each consisting of 1,000,000 trials. In each trial, N_j sensor nodes simultaneously try to join the network. The analysis is organized in two parts. Section 9.1 analyzes the case where the number of joining nodes N_j is equal to the number of available slots N_F. Then, Section 9.2 considers the case where $N_j \geq N_F$.

Parameter	Value	Parameter	Value
Data transmission rate (R)	250 Kbit/s	Association transmission time (D_{ass})	4.256 ms
Power Cons. TX mode (P_{TX})	31.32 mW	Ack transmission time (D_{ack})	352 μs
Power Cons. RX mode (P_{RX})	35.46 mW	Timeout interval (D_{to})	864 μs
Channel assessment duration (D_{CS})	128 μs	Backoff window size (W_B)	8
Data transmission time (D_{tx})	4.256 ms	Ut. pattern transmission time (D_{sap})	0.432 ms

TABLE 1: Parameters used in our analysis.

9.1 Analytical Results ($\mathbf{N_j = N_F}$)

Figure 5 shows the duration of the join process calculated using Equation 9, for $N = 10$ and different N_j values. In all the considered scenarios $N_F = N_j$. Note that analytical and simulation results are very close. As expected, the duration of the joining process significantly increases as N_j grows up. This is because, when more sensor nodes try to join the network simultaneously, the probability of collisions in the slot acquisition increases accordingly. However, in all the considered scenarios, the join procedure terminates in few superframes, as the 99-th percentile of the distribution is always less than, or equal to, 5 superframes.

Figure 6 shows the average energy consumed by all N_j joining nodes during each superframe (derived from Equation 10), for $N = 10$ and different numbers of joining nodes N_j (analytical and simulation results almost overlap). The average energy consumption exhibits the same trend for all the considered scenarios. At T_j, the energy consumption is higher than during the next superframes, as all joining nodes are still contending to acquire a slot and, hence, there is the maximum level of contention. However, the consumed energy tends to a constant value once the join procedure has been completed. As expected, the energy consumption is higher for greater values of N_j, i.e., in the presence of more joining nodes. Figure 5 shows that, even in the worst case, the 99-th percentile of the join duration is less than 6 superframes. Hence, from Figure 6 we conclude that, apart from the initial superframe T_j, the energy consumed by joining nodes is only slightly higher than the energy consumed after the join procedure has been completed (e.g., from T_{j+6} onwards). Thus, the additional energy consumed by sensor nodes during the join process is very limited, compared with the energy consumed during the entire network lifetime.

In Appendix E, we compute the overhead introduced by the join process when the centralized solution described in Section 7 is used. Figure 7 compares the energy spent by the joining nodes over time, when either JAMMY or the centralized solution is used. We can observe that, except for superframe T_j, where the contention for slot acquisition is very high, JAMMY always results in a smaller energy consumption. This is because JAMMY does not require any data exchange after the join procedure is completed.

Table 2 reports $\overline{E_{join}}$ and $\overline{E^C_{join}}$, the average total energy spent by *all* joining nodes during the entire join process, with JAMMY and the centralized solution, respectively. $\overline{E_{join}}$ and $\overline{E^C_{join}}$ characterize the total overhead of the join

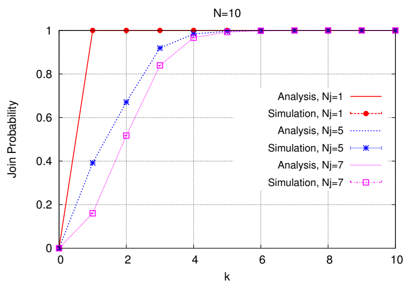

Fig. 5: $P_{join}(k)$ $(N = 10)$.

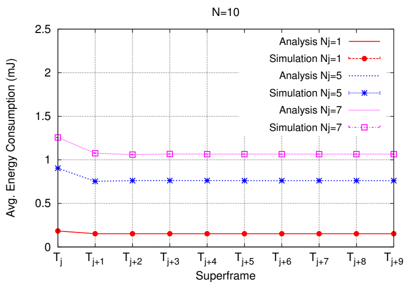

Fig. 6: $\overline{E_k}$ $(N = 10)$.

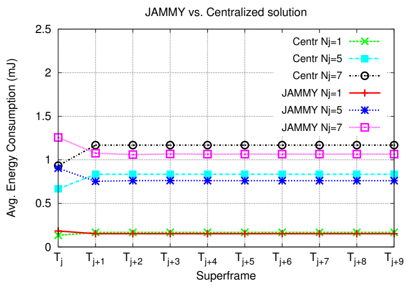

Fig. 7: $\overline{E_k}$ $(N = 10)$.

N_j	$\overline{E_{join}}$ (mJ)	$\overline{E^C_{join}}$ (mJ)	Δ_E (mJ)
1	0.18	0.13	0.05
5	1.23	0.67	0.56
7	1.85	0.93	0.91

TABLE 2: Total energy spent to complete the join process $(N = 10)$

procedure in terms of energy consumption. We compute $\overline{E_{join}}$ according to Equation 11. Also, $\overline{E^C_{join}}$ is calculated as E^C_k, with $k = 0$, according to Equation 30 (appendix D), as we optimistically assume that the joining process with the centralized solution is always completed at T_j.

From Table 2, we observe that the total energy consumption due to the join process is smaller when the centralized solution is used (see the difference $\Delta_E = \overline{E_{join}} - \overline{E^C_{join}}$ in the rightmost column). However, note that in the long term JAMMY is significantly more efficient than the centralized solution, due to its distributed nature. Specifically, in the centralized solution, the reception of the slot utilization pattern at every superframe quickly neutralizes the benefit of the shorter join process. To fix ideas, the energy Δ_E saved with the centralized solution when $N_j = 5$ is approximately equal to the energy spent by the same joining nodes to receive the slot utilization pattern for 8 consecutive superframes. That is, after only 8 superframes, the benefit due to the shorter join process is neutralized.

9.2 Simulation Results ($\mathbf{N_j \geq N_F}$)

Now, we rely on simulation to evaluate the join process overhead when the number of joining nodes is higher than or equal to the number of available slots at superframe T_j (i.e., $N_j \geq N_F$). Note that, in such a case, only N_F joining nodes can successfully acquire a slot for communication. Figure 8 shows the probability distribution function of the joining time, when $N = 10$, $N_A = N_F = 5$, and different values of N_j are considered. In all the experiments, we considered the join process completed when N_F out of N_j joining nodes have successfully acquired a slot.

As shown in Figure 8, the duration of the join process increases only slightly with the number of joining nodes N_j. Specifically, the 99-th percentile of the distribution is always less than, or equal to, 5 superframes. This means that JAMMY is very efficient in managing contention, even in the presence of a large number of joining nodes. We also measured the energy spent by joining nodes that successfully acquire a slot. We did not observe a significant difference between the cases $N_j = N_F$ and $N_j \geq N_F$.

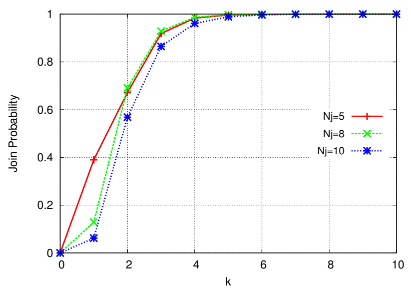

Fig. 8: $P_{join}(k)$ $(N = 10,\ N_A = 5)$.

10 Conclusion

We have presented JAMMY, a novel and distributed solution to selective jamming attacks in TDMA WSNs. JAMMY forces the adversary to perform the attack at random, hence reducing its effectiveness to $1/N$, where N is the number of slots in the superframe. We have evaluated JAMMY through analysis, simulation and measurements in a real large-scale testbed. When the network is in steady-state conditions, JAMMY introduces no communication or energy overhead, regardless the number of sensor nodes in the network. Hence, it outperforms a generic centralized solution operating in similar conditions, in terms of available bandwidth and energy efficiency. We have analyzed JAMMY in dynamic conditions, i.e., when nodes join or leave. Our results show that the join procedure always terminates in a short number of superframes, and introduces a limited energy consumption on joining nodes. Future works will focus on extending JAMMY for multichannel TDMA WSNs (e.g., with reference to IEEE 802.15.4e).

Acknowledgment

The authors sincerely thank the anonymous referees and the associate editor for their insightful comments and suggestions. This work was supported by NSF (CNS-1404677, IIS-1404673, DGE-1433659, CNS-1355505), European Commission (FP7 Project SEGRID), Italian Ministry of Education, University and Research (PRIN Project TENACE), and University of Pisa (PRA 2015 program).

References

[1] A. D. Wood, J. A. Stankovic and G. Zhou. DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks. In *Proceedings of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 60–69. IEEE Computer Society, June 2007.

[2] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[3] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos and G. Pantziou. A Survey on Jamming Attacks and Countermeasures in WSNs. *IEEE Communications Surveys Tutorials*, 11(4):42–56, 2009.

[4] A. Proaño and L. Lazos. Packet-Hiding Methods for Preventing Selective Jamming Attacks. *IEEE Transactions on Dependable and Secure Computing*, 9(1):101–114, January/February 2012.

[5] C. Paar and J. Pelzl. *Understanding Cryptography*. Springer, 2010.

[6] D. De Guglielmo, G. Anastasi and M. Conti. A Localized Slot Allocation Algorithm for Wireless Sensor Networks. In *Proceedings of the 12th IEEE Annual Mediterranean Ad Hoc Networking Workshop*, pages 89–96. IEEE Computer Society, June 2013.

[7] D. R. Raymond and S.F. Midkiff. Denial-of-Service in Wireless Sensor Networks: Attacks and Defenses. *IEEE Pervasive Computing*, 7(1):74–81, January-March 2008.

[8] M. Doddavenkatappa, M. Chan, and A. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 302–316. Springer, 2012.

[9] A. Dunkels, B. Grönvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004.

[10] E. Cole. *Network Security Bible, 2nd Edition*. Wiley, 2009.

[11] F. Ashraf, Y.-C. Hu and R.H. Kravets. Bankrupting the jammer in WSN. In *Proceedings of the IEEE 9th International Conference on Mobile Adhoc and Sensor Systems*, pages 317–325, October 2012.

[12] G. Dini and I. M. Savino. LARK: A Lightweight Authenticated ReKeying Scheme for Clustered Wireless Sensor Networks. *ACM Transactions on Embedded Computing Systems*, 10(4), 2011.

[13] G. Dini and M. Tiloca. HISS: a HIghly Scalable Scheme for group rekeying. *The Computer Journal*, 56(4):508–525, November 2013.

[14] H. Mustafa, X. Zhang, Z. Liu, W. Xu and A. Perrig. Jamming-Resilient Multipath Routing. *IEEE Transactions on Dependable and Secure Computing*, 9(6):852–864, November 2012.

[15] IEEE Computer Society. *IEEE Standard for Local and Metropolitan Area Networks, Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, September 2011.

[16] J. Schiller and S. Crocker. *Randomness Requirements for Security*. Internet Engineering Task Force, Fremont, CA, USA, 2005.

[17] K. Birman. *Guide to Reliable Distributed Systems. Building High-Assurance Applications and Cloud-Hosted Services*. Springer, 2012.

[18] K. Pelechrinis, C. Koufogiannakis and S. V. Krishnamurthy. Gaming the Jammer: Is Frequency Hopping Effective? In *Proceedings of the 7th int. conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 187–196. IEEE Press, June 2009.

[19] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd Edition*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[20] L. Lazos, S. Liu and M. Krunz. Mitigating Control-channel Jamming Attacks in Multi-channel Ad Hoc Networks. In *Proceedings of the second ACM conference on Wireless network security*, pages 169–180. ACM, March 2009.

[21] M. Cagalj, S. Capkun and J.-P. Hubaux. Wormhole-Based Antijamming Techniques in Sensor Networks. *IEEE Transactions on Mobile Computing*, pages 100–114, January 2007.

[22] M. Tiloca, D. De Guglielmo, G. Dini and G. Anastasi. SAD-SJ: a Self-Adaptive Decentralized Solution Against Selective Jamming Attack in Wireless Sensor Networks. In *Proceedings of the 18th IEEE International Conference on Emerging Technology & Factory Automation*, pages 1–8. IEEE Computer Society, September 2013.

[23] J. Misra and D. Gries. A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.

[24] R. Daidone, G. Dini and M. Tiloca. On Experimentally Evaluating the Impact of Security on IEEE 802.15.4 Networks. In *Proceedings of International Conference on Distributed Computing in Sensor Systems and Workshops*, pages 1–6, June 2011.

[25] R. Daidone, G. Dini and M. Tiloca. A Solution to the GTS-based Selective Jamming Attack on IEEE 802.15.4 Networks. *Wireless Networks*, 20(5):1223–1235, November 2013.

[26] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd Edition*. Wiley, 2008.

[27] R. L. Pickholtz, D. L. Schilling and L. B. Milstein. Theory of Spread-Spectrum Communications - A Tutorial. *IEEE Transactions on Communications*, 30(5):855–884, May 1982.

[28] R. Sokullu, I. Korkmaz and O. Dagdeviren. GTS Attack: An IEEE 802.15.4 MAC Layer Attack in Wireless Sensor Networks. *Int.l Journal On Advances in Internet Technologies*, 2(1):104–114, 2009.

[29] S. Rafaeli and D. Hutchison. A Survey of Key Management for Secure Group Communication. *ACM Computing Surveys*, 35(3):309–329, September 2003.

[30] S. Stojanovski and A. Kulakov. Efficient Attacks in Industrial Wireless Sensor Networks. In *ICT Innovations 2014*, volume 311 of *Advances in Intelligent Systems and Computing*, pages 289–298. Springer International Publishing, 2015.

[31] T. Verhoeff. Reward Variance in Markov Chains: A Calculational Approach. In *Proceedings of Eindhoven FASTAR Days 2004*. Technische Universiteit Eindhoven, September 2004.

[32] Texas Instruments. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee ready RF Transceiver. http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2012.

[33] W. Xu, K. Ma, W. Trappe and Y. Zhang. Jamming Sensor Networks: Attack and Defense Strategies. *IEEE Network*, 20(3):41–47, 2006.

[34] W. Xu, T. Wood, W. Trappe and Y. Zhang. Channel Surfing and Spatial Retreats: Defenses Against Wireless Denial of Service. In *Proceedings of the 3rd ACM Workshop on Wireless Security*, WiSe '04, pages 80–89. ACM, October 2004.

[35] W. Xu, W. Trappe and Y. Zhang. Channel Surfing: Defending Wireless Sensor Networks from Interference. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, pages 499–508. ACM, April 2007.

[36] W. Xu, W. Trappe, Y. Zhang and T. Wood. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *Proceedings of the 6th ACM International Symposium on Mobile ad hoc Networking and Computing*, pages 46–57. ACM, May 2005.

[37] Z. Lu, W. Wang and C. Wang. Modeling, Evaluation and Detection of Jamming Attacks in Time-Critical Wireless Applications. *IEEE Transactions on Mobile Computing*, 13(8):1746–1759, August 2014.

Marco Tiloca is a Senior Researcher at SICS Swedish ICT AB, Sweden. His research interests include network and communication security, Internet of Things, key management and attack simulation.

Domenico De Guglielmo is a Postdoctoral Researcher in the Dept. of Information Engineering at the University of Pisa. His research interests are in the field of WSNs and Internet of Things.

Gianluca Dini is an Associate Professor in the Dept. of Information Engineering at the University of Pisa. His research interests are in the field of distributed computing systems, with particular reference to security. He has published 100+ papers and has participated in many projects funded by the Commission of the European Community, the Italian Government and private companies.

Giuseppe Anastasi is a Full Professor at the Dept. of Information Engineering at the University of Pisa, Italy. He is also the Director of the CINI National Smart Cities Lab. His research interests include pervasive computing, sensor networks, sustainable computing, and ICT or smart cities. He has contributed to many research programs funded by both national and international institutions. He has co-edited two books and published more than 120 papers in international journal and conferences. Dr. Anastasi is an Associate Editor of Sustainable Computing, and Pervasive and Mobile Computing.

Sajal K. Das is the Chair of Computer Science Department and Daniel St. Clair Endowed Chair at the Missouri University of Science and Technology. During 2008-2011, he served the US National Science Foundation as a Program Director in the Computer Networks and Systems Division. During 1999-2013 he was a University Distinguished Scholar Professor of Computer Science and Engineering, and founding director of Center for Research in Wireless Mobility and Networking (CReWMaN) at the University of Texas at Arlington. His current research interests include wireless and sensor networks, mobile and pervasive computing, cyber-physical systems and smart environments including smart health care and smart grid, security and privacy, distributed and cloud computing, biological and social networks, applied graph theory and game theory. He has published over 600 papers in high quality journals and peer-reviewed conferences, and 51 book chapters. He has also coauthored four books and holds 5 US patents. Dr. Das is a recipient of the IEEE Computer Society Technical Achievement Award for pioneering contributions in sensor networks and mobile computing. He is the Founding Editor-in-Chief of Elsevier's Pervasive and Mobile Computing (PMC) journal, and an Associate Editor of IEEE Transactions on Mobile Computing, ACM Transactions on Sensor Networks, ACM/Springer Wireless Networks, Journal of Parallel and Distributed Computing, and Journal of Peer-to-Peer Networking and Applications. He is a Fellow of the IEEE.

APPENDIX

A. Optimized random permutation

In Section 4, we considered the Knuth shuffle algorithm to perform a random permutation of slots assigned to active sensor nodes at each superframe. Here we present an optimized permutation function for transmitter-only nodes, namely *permute_no_vec()*. As shown in Algorithm 5, a generic node u, currently using the i-th slot in the superframe to transmit data to its parent node, invokes *permute_no_vec()* providing i as input argument. The function returns the index of the slot to be used in the next superframe. It does not rely on any actual vector, thus saving a considerable amount of memory. This is extremely important when dealing with resource constrained devices such as sensor nodes.

```
int permute_no_vec(unsigned old_index){
  unsigned i;
  unsigned n;
  unsigned new_index = old_index;
  for (i = N - 1; i >= 0; i--) {
    n = random() % (i + 1);
    if (i == new_index)
      new_index = n;
    else if (n == new_index)
       new_index = i;
    // else new_index = new_index;
  }
  return new_index;
}
```

Algorithm 5: *permute_no_vec* function.

B. Proof of Claim 1

Claim 1. *The SSP algorithm maintains the Uniqueness and Collision-Free properties at each superframe.*

Proof: As to the Uniqueness Property, the SSP algorithm simply permutes the permutation vector elements. So only one element of the resulting permutation vector contains the value 1 and thus the property is maintained.

As to the Collision-Free Property, let us define $\mathcal{L} = (L_1, \ldots, L_N)$ where L_i is the set of non-interfering links that use slot s_i in the current superframe. Consider a link $(u,t) \in L_i$ such that, to fix ideas, $v_u[i] = 1$ and $v_t[i] = 2$. Let Π be the same permutation that u and t compute at the end of the current superframe, and v'_u and v'_t be the vectors resulting from applying Π to v_u and v_t, respectively. Finally, assume that Π maps the i-th element into the j-th. It follows that $v'_u[j] = 1$ and $v'_t[j] = 2$, that is, link (u,t) becomes active during slot s_j in the next superframe. As all nodes compute the same permutation Π, then all links using slot s_i during the current superframe will be active during slot s_j in the next superframe. This means that L_i is permuted into L_j. More generally, this means that permutation Π randomly permutes elements in $\mathcal{L}$. As a consequence, every slot of the superfame becomes associated with a different set of non-interfering links. Therefore, the Collision-Free Property is maintained. □

C. Proofs of Claims 2 and 3

Claim 2.

$$P_s^F(M) = M \cdot \sum_{w=0}^{W_B-1} \left(\frac{1}{W_B}\right) \cdot \left(\frac{W_B - 1 - w}{W_B}\right)^{M-1} \tag{3}$$

$$P_c^F(k \mid M) = \binom{M}{k} \sum_{w=0}^{W_B-1} \left(\frac{1}{W_B}\right)^k \cdot \left(\frac{W_B - 1 - w}{W_B}\right)^{M-k} \tag{4}$$

$$P_b^F(h \mid M) = P_c^F(M - h \mid M) \tag{5}$$

Proof: Let us first consider Equation 3. A successful transmission occurs when one joining node generates a backoff time shorter than the one of all the other $(M-1)$ contending nodes. Given W_B the backoff window size, every node can extract a backoff $w \in \{0,\ 1,\ ...,\ W_B - 1\} \cdot D_{bo}$. Hence, Equation 3 can be explained as follows. For every possible backoff time w that can be generated by a joining node with probability $\frac{1}{W_B}$, the second term inside the sum is the probability that the remaining $(M-1)$ sensor nodes extract a backoff time larger than w. Then, all M combinations, corresponding to the different sensor nodes, are considered.

Let us now consider Equation 4, i.e. the case where two or more joining nodes generate the same backoff time, thus starting their transmission at the same time and experiencing collision. In Equation 4, for every possible backoff time w, the term inside the sum gives the probability that k joining nodes randomly pick up a value equal to w, and $(M-k)$ sensor nodes choose a value larger than w. Of course, all $\binom{M}{k}$ possible combinations are considered.

Finally, we consider Equation 5, i.e. the case when $h = (M-k)$ nodes find the channel busy. Since h nodes have found the channel busy, $(M-h)$ nodes have extracted the same (minimum) backoff value, and collided. Therefore, $P_b^F(h \mid M) = P_c^F(M-h \mid M)$. □

Claim 3.

$$E_s(M) = M \cdot P_{RX} \cdot D_{CS} + P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{ack} \tag{6}$$

$$E_c(k \mid M) = M \cdot P_{RX} \cdot D_{CS} + k \cdot (P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{to}) \tag{7}$$

$$E_u = P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{ack} \tag{8}$$

Proof: Let us focus on Equation 6 that provides the total energy spent by nodes when one of the M joining nodes wins the contention. The first term in Equation 6 accounts for the energy consumed by all the M joining nodes to perform their channel sensing operation, while the other terms account for the additional energy consumed by the winner node for transmitting the packet $(P_{TX} \cdot D_{tx})$ and receiving the ACK $(P_{RX} \cdot D_{ack})$.

Equation 7 is derived by following the same line of reasoning. Hence, we omit its description. Finally, let us focus on Equation 8 that provides the energy E_u spent by a sensor node during an acquired slot. E_u is derived accounting for both the energy due to the transmission of the data packet and the energy spent to receive the ACK packet. □

D. Derivation of the Discrete Time Markov Chain

In this section, we describe the derivation of the Discrete Time Markov Chain (DTMC) that models the JAMMY join procedure. First, we derive the DTMC in a simple scenario. Specifically, we consider the case when the superframe is composed of $N = 3$ slots, 1 slot is acquired (i.e. $N_A = 1$), and $N_j = 2$ nodes start their join procedure at T_j. Then, in Section D.2, we derive the DTMC for arbitrary values of N, N_A, and N_j.

D.1 Markov Chain Derivation in a Simple Scenario

With reference to the considered example scenario, the initial system state is $X_j = [2,\ 0,\ 0]$, i.e., the two joining nodes start contending for the first slot of superframe T_j (see Figure 9). Then, the system can evolve in different states, depending on both the slot utilization pattern at superframe T_j, i.e., S_j, and the specific events that occur during the contention for each slot in T_j. To properly model the evolution of the system we have to consider all possible slot utilization patterns S_j at superframe T_j. Since there is a single acquired slot at T_j, the possible slot utilization patterns are $S_j^1 = [A,\ F,\ F]$, $S_j^2 = [F,\ A,\ F]$ and $S_j^3 = [F,\ F,\ A]$ where $F(A)$ indicates a free (acquired) slot. Also, since the different slot utilization patterns are due to the SSP algorithm (Algorithm 2), all patterns have the same probability to occur which implies $P(S_j^1) = P(S_j^2) = P(S_j^3) = \frac{1}{3}$.

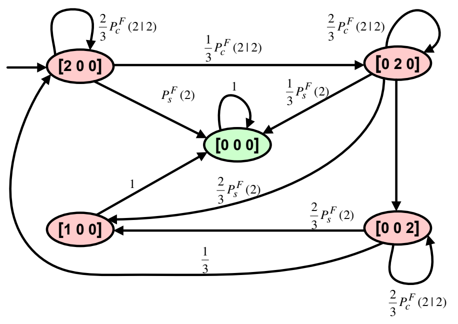

Fig. 9: Markov chain for $N = 3$, $N_A = 1$, $N_j = 2$.

Let us assume $S_j = S_j^1 = [A,\ F,\ F]$. In this case, the two joining nodes contend for the first slot of the superframe, which is already used by an active node. Hence, they lose contention and, according to Algorithm 3, they have to retry at the second slot. Then, there are two possible evolutions for the system, depending on the specific event that occurs at the second slot. In case of a collision between the two sensor nodes, which occurs with a probability equal to $P_c^F(2 \mid 2)$, the system at superframe T_{j+1} will be in state $[0,\ 2,\ 0]$, since the two nodes will retry to contend for the second slot (Assumption 3). Instead, if one of the two joining nodes wins the contention at the second slot, which occurs with probability $P_s^F(2)$, it becomes the owner of the slot and ends the join procedure. Hence, starting from superframe T_{j+1}, it has to be considered as an active node. The sensor node which loses the contention at the second slot, retries to contend for the third slot. Since we are assuming $S_j = [A,\ F,\ F]$, the third slot is free. Hence, it surely wins the contention and ends the join procedure. Thus, the system state, at T_{j+1}, will be $[0,\ 0,\ 0]$, i.e., all joining nodes have found their slot and have completed the join procedure.

Let us now consider the case $S_j = S_j^2 = [F,\ A,\ F]$. In this case there are two possible transitions for the system. The state of the system could remain the same, i.e., $X_{j+1} = X_j = [2,\ 0,\ 0]$, or the system could evolve in state $X_{j+1} = [0,\ 0,\ 0]$. Since the first slot is free in this case, the first transition occurs when the two joining nodes experience a collision at the first slot, which happens with probability $P_c^F(2 \mid 2)$. Instead, the second transition happens when a success occurs in the first slot with probability $P_s^F(2)$.

Finally, let us analyze the last case, i.e. $S_j = S_j^3 = [F,\ F,\ A]$. The possible transitions for the system are the same as in the previous case implying the system could transit to state $[2,\ 0,\ 0]$, with probability $P_c^F(2 \mid 2)$, or to state $[0,\ 0,\ 0]$ with probability $P_s^F(2)$. Figure 9 reports all transitions of the system starting from state $[2,\ 0,\ 0]$, along with their probability. Note that the probability of each transition also considers the probability of each possible slot utilization pattern to occur.

Following a similar line of reasoning, we can derive all possible transitions that can occur when the system is initially in state $[0,\ 2,\ 0]$. When the network is in this state, there is only one acquired slot in the network, since all the two joining nodes have not terminated their join procedure yet. Hence, the possible equiprobable slot utilization patterns S_m are $S_m = [A,\ F,\ F]$, $[F,\ A,\ F]$, $[F,\ F,\ A]$. As shown in Figure 9, there are four possible transitions in this case, namely $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 0]$, $[0,\ 2,\ 0] \rightarrow [0,\ 2,\ 0]$, $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 2]$, and $[0,\ 2,\ 0] \rightarrow [1,\ 0,\ 0]$. First, we analyze the transition $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 0]$. This transition occurs only when the second and third slots are free, i.e., $S_m = [A,\ F,\ F]$, and one of the two joining nodes wins the contention with the other one at the second slot, which happens with probability equal to $P_s^F(2)$. Hence, the probability for transition $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 0]$ to occur is given by $\frac{1}{3} \cdot P_s^F(2)$. Transition $[0,\ 2,\ 0] \rightarrow [0,\ 2,\ 0]$ occurs when the second slot is free, i.e., $S_m = [A,\ F,\ F]$ or $[F,\ F,\ A]$, and a collision between the two joining nodes occurs. Hence, this transition has a probability equal to $\frac{2}{3} \cdot P_c^F(2 \mid 2)$ to occur. Transition $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 2]$ occurs when the two joining nodes move from the second to the third slot and, then, they experience a collision. This is possible only if the second slot is acquired by an active link, i.e., $S_m = [F,\ A,\ F]$. Hence the transition has a probability equal to $\frac{1}{3} \cdot P_c^F(2 \mid 2)$.

Finally, transition $[0,\ 2,\ 0] \rightarrow [1,\ 0,\ 0]$ can occur in two cases. The first one is when the second slot is acquired by an active link, i.e., $S_m = [F,\ A,\ F]$, the two joining nodes move to the third slot, and one of them wins the contention for the third slot. The sensor node winning the contention terminates the join procedure, acquires a slot and becomes an active node. Instead, the sensor node losing the contention retries to contend at the first slot of the subsequent superframe. These events occur with probability $\frac{1}{3} \cdot P_s^F(2)$. The second situation which leads the system to state $[1,\ 0,\ 0]$ is when the second slot is free and the third one is acquired by any active link, i.e., $S_m = [F,\ F,\ A]$, and a success occurs at the second slot. In this case, one joining node wins the contention and becomes an active node while the other one tries to contend for the third slot. Since the third slot is already acquired, the joining node finds the channel busy and schedules a retry at the first slot of the subsequent superframe.

Let us now analyze the transitions originating from state $[0,\ 0,\ 2]$. There are three possible transitions in this case, namely $[0,\ 0,\ 2] \rightarrow [0,\ 0,\ 2]$, $[0,\ 0,\ 2] \rightarrow [2,\ 0,\ 0]$, and $[0,\ 0,\ 2] \rightarrow [1,\ 0,\ 0]$. The first transition occurs when the third slot is free, i.e., $S_m = [A,\ F,\ F]$ or $S_m = [F,\ A,\ F]$ and a collision between the two joining nodes occurs at the third slot. The corresponding probability is $\frac{2}{3} \cdot P_c^F(2 \mid 2)$. Transition $[0,\ 0,\ 2] \rightarrow [2,\ 0,\ 0]$ is possible only when the third slot is acquired. In fact, in this case, the two joining nodes find the channel busy at the third slot and retry to the first slot of the subsequent superframe. This event has a probability equal to $\frac{1}{3}$ to occur. The last transition is $[0,\ 0,\ 2] \rightarrow [1,\ 0,\ 0]$. There are two conditions for this transition to occur. First, the third slot has to be free, i.e., $S_m = [A,\ F,\ F]$ or $S_m = [F,\ A,\ F]$. Second, a success has to occur at the third slot. Hence, the transition probability is equal to $\frac{2}{3} \cdot P_s^F(2)$.

Let us turn our attention to state $[1,\ 0,\ 0]$. In this case one of the two joining nodes has completed the join procedure in a previous superframe. Hence, the possible slot utilization patterns are $S_m = [A,\ A,\ F]$, $[A,\ F,\ A]$, and $[F,\ A,\ A]$. There is only one possible transition for the system starting from state $[1,\ 0,\ 0]$. In fact, whatever is the slot utilization pattern, the remaining joining node will surely acquire a free slot in the current superframe. Hence, the system evolves to state $[0,\ 0,\ 0]$ with probability equal to 1. Finally, when the system reaches state $[0,\ 0,\ 0]$, the join procedure is completed. Hence, the system will never change its state; in other words state $[0,\ 0,\ 0]$ is an absorbing state.

Each transition described above is characterized by an average energy consumed by joining sensor nodes. Figure 10 shows the average energy consumption associated with each transition derived according to Equations 6 and 7. For the sake of simplicity, we only explain the computation of the average energy consumption for transitions originating from the inital state of the system, $[2,\ 0,\ 0] \rightarrow [2,\ 0,\ 0]$, $[2,\ 0,\ 0] \rightarrow [0,\ 2,\ 0]$, and $[2,\ 0,\ 0] \rightarrow [0,\ 0,\ 0]$. The average energy spent during all the other transitions can be derived following the same line of reasoning. Let us focus on the transition $[2,\ 0,\ 0] \rightarrow [2,\ 0,\ 0]$. As mentioned above, this transition occurs when the two joining nodes experience a collision at the first slot of the superframe. Hence, according to Equation 7, the average energy consumption is equal to $E_c(2 \mid 2)$ in this case. Transition $[2,\ 0,\ 0] \ \rightarrow\ [0,\ 2,\ 0]$ occurs when: i) the two joining nodes find the channel busy during the first slot and move to the second slot, and ii) they experience a collision. During the first slot the two nodes perform a channel sensing operation. Hence, they spend an energy equal to $2 \cdot E_{CS}$. Instead, the energy consumed due to the collision at the second slot is equal to $E_c(2 \mid 2)$. Thus, the energy consumption for this transition is $2 \cdot E_{CS} + E_c(2 \mid 2)$. The last transition we consider is $[2,\ 0,\ 0] \ \rightarrow\ [0,\ 0,\ 0]$ which occurs when both joining nodes succeed in acquiring a free slot during the superframe.

There are three different cases that cause this transition. The first one happens when i) $S_m = [A,\ F,\ F]$ and ii) two successes occur, one at the second slot and one during the third slot. The energy spent in this case is equal to $2 \cdot E_{CS} + E_s(2) + E_s(1)$ since the two nodes sense the channel during the first slot and there are two successful events. Also, the probability for this case to occur is equal to $\frac{1}{3} \cdot P_s^F(2)$. The second case is when $S_m = [F,\ A,\ F]$ and two successful transmissions occur, one at the first slot and one at the third slot. Hence, the energy spent by the joining nodes is equal to $E_s(2) + E_{CS} + E_s(1)$. The probability of this case to occur is $\frac{1}{3} \cdot P_s^F(2)$. The last case happens when $S_m = [F,\ F,\ A]$ and two successes occur, one at the first and one at the second slot. The energy spent in this case is equal to $E_s(2) + E_s(1)$ while the probability for the case to occur is $\frac{1}{3} \cdot P_s^F(2)$. As it can be observed all the three cases have the same probability to occur. Hence, we can calculate the average energy consumption simply by performing an arithmetic mean of the energies spent in the three different cases. Thus, the average energy consumption for this transition is equal to $E_s(2) + E_s(1) + E_{CS}$.

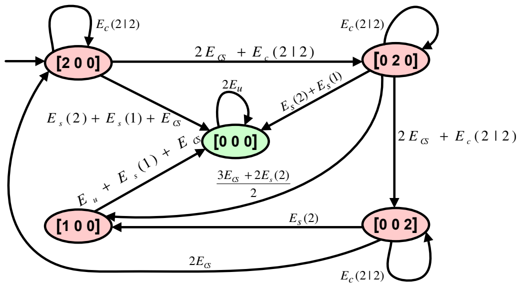

Fig. 10: Energy consumptions for $N = 3$, $N_A = 1$, $N_j = 2$.

D.2 Markov Chain Derivation in the General Case

In the previous section, we derived a DTMC for the simple case when the superframe consists of $N = 3$ slots. In this section, we derive the DTMC model in the general case - for any values of N, N_A, N_j. Specifically, we first define a *generate_states()* function, used to generate all possible states of the DTMC. To this end, we refer to the event probabilities derived in Section 8.1. Then, we rely on *generate_states()* to derive the DTMC.

We recall that we observe the system at the beginning of every superframe T_m. Specifically, we represent the system state as a vector $X_m = [n_1,\ n_2, ...,\ n_N]$, where element n_i refers to the i-th slot, namely s_i, of T_m. Specifically, n_i indicates the number of joining nodes that directly try contention at slot s_i. Since we consider the presence of N_j joining nodes, we have $n_i \leq N_j, \forall\ i$. In the following, we refer to T_j as the superframe at which joining nodes start to execute the slot acquisition process, and N_A as the number of acquired slots at the beginning of superframe T_j. Also, we define $S_m = [\overline{s}_1,\ \overline{s}_2,\ ...,\ \overline{s}_N]$ as the slot utilization pattern at the beginning of a given superframe T_m. Specifically, $\overline{s}_i$ indicates the status of slot s_i, i.e., if it is either *free* (F) or *acquired* (A). Of course, superframe T_j features N_A slots whose status is A.

Initially, the system state is $X_j = [N_j,\ 0, ...,\ 0]$, implying all joining nodes have scheduled their transmission at the first slot of superframe T_j. Then, the system can evolve in different states, depending on both the slot utilization pattern S_j and the specific events that occur during the contention for each slot.

State Generation

Algorithm 6 provides details of the function $generate\ states\ (X_m, S_m)$. Such a function takes two parameters as input (line 1), namely X_m and S_m, that represent the network state and the slot utilization pattern, at the beginning of superframe T_m, respectively. It returns a set $G_{X_m S_m}$ of $transition$ vectors, each of which includes the following three fields:

• $X_{m+1} = [n_1,\ n_2, ...,\ n_N]$ represents one of the possible states where the network can evolve to from superframe T_m to superframe T_{m+1}, starting from state X_m and slot utilization pattern S_m.

• $p_{X_m X_{m+1}}$ is the probability that the system changes its state from X_m to X_{m+1}, given the S_m slot utilization pattern at superframe T_m.

• $e_{X_m X_{m+1}}$ is the energy consumed by all joining nodes when the system changes its state from X_m to X_{m+1}, given the S_m slot utilization pattern at superframe T_m.

In practice, the function $generate\_states()$ produces $G_{X_m S_m}$, i.e., the set of all possible states X_{m+1} where the system can evolve to from state X_m and slot utilization pattern S_m. For each X_{m+1}, the transition probability $p_{X_m X_{m+1}}$ and the energy consumption $e_{X_m X_{m+1}}$ are also reported.

Initialization. The function $generate\_states()$ relies on an iterative approach to generate all possible $transition$ vectors. First of all, $generate\_states()$ creates and initializes transition $init$ (lines 2-7). Specifically, $init.X_{m+1}$ is set to X_m, while $init.p_{X_m X_{m+1}}$ is set to 1, since no events have been considered yet. Instead, $init.e_{X_m X_{m+1}}$ assumes an initial value equal to $E_u \cdot (N_j - \sum_{i=1}^{N} X_m[i])$, where E_u is the energy spent by joining nodes that have already completed their join procedure. Since there are $N_j = N_F$ joining nodes at superframe T_j , the number of joining nodes that have already completed their join procedure can be calculated as $N_j - \sum_{i=1}^{N} X_m[i]$. Then, transition $init$ is added to the set Ω_0. In general, Ω_i for $1 \leq i \leq N$, represents the set of $transitions$ generated by $generate\_states()$ after examining i slots in the current superframe. During slot s_i for $1 \leq i \leq N$, the function $generate\_states()$ considers all transitions $\omega \in \Omega_{i-1}$ (lines 8-9). In case no joining nodes are contending slot s_i, the current transition is skipped, and the next one is

```
1:  $G_{X_m S_m}$ generate_states $(X_m, S_m)$
2:  $transition\ init$    //Initialization
3:  $init.X_{m+1} = X_m$
4:  $init.p_{X_m X_{m+1}} = 1$
5:  $init.e_{X_m X_{m+1}} = E_u \cdot (N_j - \sum_{i=1}^{N} X_m[i])$
6:  $\Omega_0 = \emptyset$
7:  $\Omega_0 = \Omega_0 \cup init$
8:   for $i$ in $1\ ...\ N$
9:     for each $\omega \in \Omega_{i-1}$
10:      if $\omega.X_{m+1}[i] = 0$ then
11:       $transition\ gen = \omega$
12:       $\Omega_i = \Omega_i \cup gen$
13:       continue
14:      end if
15:      if $\overline{s}_i \in S_m = A$ then    // Acquired slot
16:       $transition\ gen = \omega$
17:       $gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + \omega.X_{m+1}[i]$
18:       $gen.X_{m+1}[i] = 0$
19:       $gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot 1$
20:       $gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + \omega.X_{m+1}[i] \cdot E_{CS}$
21:       $\Omega_i = \Omega_i \cup gen$
22:      end if
23:      if $\overline{s}_i \in S_m = F$ then    // Free slot
24:       $transition\ gen = \omega$    // Success
25:       $gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + \omega.X_{m+1}[i] - 1$
26:       $gen.X_{m+1}[i] = 0$
27:       $gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_s^F(\omega.X_{m+1}[i])$
28:       $gen.e_{X_m X_{m+1}} = \omega.e_{X_m Y_{m+1}} + E_s(\omega.X_{m+1}[i])$
29:       $\Omega_i = \Omega_i \cup gen$
30:       for $k = 2$ to $\omega.X_{m+1}[i]$    // Collision
31:        $transition\ gen = \omega$
32:        $gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + (\omega.X_{m+1}[i] - k)$
33:        $gen.X_{m+1}[i] = k$
34:        $gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_c^F(k \mid \omega.X_{m+1}[i])$
35:        $gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + E_c(k \mid \omega.X_{m+1}[i])$
36:        $\Omega_i = \Omega_i \cup gen$
37:       end for
38:      end if
39:     end for
40:   end for
41:  return $G_{X_m S_m} = \Omega_N$
42: end function
```

Algorithm 6: Function *generate_states()*

considered (lines 10-14). Otherwise, it behaves differently depending on whether slot s_i is *acquired* (A) or *free* (F).

Acquired Slot. First, let us consider the case when slot s_i is acquired (lines 15-22). In such a case, all nodes contending for slot s_i find the channel busy while performing their carrier sense operation and, thus, schedule a retry at the next slot s_{i+1}. Then, for each transition $\omega \in \Omega_{i-1}$, a new transition $gen = \omega$ is created (line 16), and the following operations are performed (lines 17-21).
1) All the $\omega.X_{m+1}[i]$ nodes contending for slot s_i retry at the next slot s_{i+1}, during which other $\omega.X_{m+1}[(i+1)\%N]$ nodes are going to contend. Therefore

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + \omega.X_{m+1}[i] \quad (13)$$

2) No nodes remain on slot s_i, then

$$gen.X_{m+1}[i] = 0 \quad (14)$$

3) All contending nodes retry at the next slot for sure, hence the overall transition probability does not change, that is

$$gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot 1 \quad (15)$$

4) All the $\omega.X_{m+1}[i]$ nodes perform a channel sensing during slot s_i. Hence, the overall energy consumed by joining nodes during slot s_i is equal to $\omega.X_{m+1}[i] \cdot E_{CS}$. More formally, we have

$$gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + \omega.X_{m+1}[i] \cdot E_{CS} \quad (16)$$

5) Finally, being a possible transition state generated at slot s_i, gen is added to the set Ω_i, that is

$$\Omega_i = \Omega_i \cup gen \quad (17)$$

Free Slot. Now we consider the case when slot s_i is *free* (F) (lines 23-39). In such a case, different kinds of events can occur during the contention for slot s_i. Specifically, for each $\omega \in \Omega_{i-1}$, the algorithm considers all the events that can occur at slot s_i in the presence of $\omega.X_{m+1}[i]$ sensor nodes trying to access it. Thus, there are exactly $\omega.X_{m+1}[i]$ events to be considered, that can be classified into two different classes, namely *success* and *defeat*. The former regards the cases when only one sensor node wins the contention and successfully transmits its packet at slot s_i, while all other sensor nodes find the channel busy. The latter refers to the cases when two or more sensor nodes experience a collision at slot s_i, while the remaining ones find the channel busy.

Success. First, let us focus on the case when one of the $\omega.X_{m+1}[i]$ contending sensor nodes successfully transmits its packet (lines 24-29). Then, a new transition $gen = \omega$ is created, and the following operations are performed.
1) All nodes contending slot s_i but one (i.e., $\omega.X_{m+1}[i] - 1$), retry at the next slot s_{i+1}, during which other $\omega.X_{m+1}[(i+1)\%N]$ nodes are going to contend. Hence,

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + (\omega.X_{m+1}[i] - 1) \quad (18)$$

2) The $\omega.X_{m+1}[i] - 1$ nodes that have lost the contention at slot s_i, retry at the next slot s_{i+1}, whereas the winner node terminates its own join procedure. Thus, no joining nodes remain in slot s_i, then

$$gen.X_{m+1}[i] = 0 \quad (19)$$

3) We need to consider both i) the probabilities of all events occurred before slot s_i, which is $\omega.p_{X_m X_{m+1}}$; and ii) the probability of the specific event happened during slot s_i, which is $P_s^F(\omega.X_{m+1}[i])$. Thus,

$$gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_s^F(\omega.X_{m+1}[i]) \quad (20)$$

4) In the presence of $\omega.X_{m+1}[i]$ contending nodes, the overall energy consumed by joining nodes is equal to $E_s(\omega.X_{m+1}[i])$. Hence, we have

$$gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + E_s(\omega.X_{m+1}[i]) \quad (21)$$

5) Finally, being a possible transition state generated at slot s_i, we add gen to the set Ω_i such that

$$\Omega_i = \Omega_i \cup gen \quad (22)$$

Collision. Now we consider the case when two ore more sensor nodes experience a collision at slot s_i (lines 30-37). For each possible number of colliding nodes k, $2 \leq k \leq \omega.X_{m+1}[i]$, a new transition $gen = \omega$ is created, and the following operations are performed.
1) All nodes that have found the medium busy, i.e. $\omega.X_{m+1}[i] - k$, retry at the next slot s_{i+1} (according to Algorithm 3), during which other $\omega.X_{m+1}[(i+1)\%N]$ nodes are going to contend. Hence,

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + (\omega.X_{m+1}[i] - k) \quad (23)$$

2) All the k colliding nodes remain in slot s_i, then

$$gen.X_{m+1}[i] = k \quad (24)$$

3) We have to consider both i) the probabilities of all events occurred before slot s_i, i.e. $\omega.p_{X_m X_{m+1}}$; and ii) the probability that k sensor nodes out of the $\omega.X_{m+1}[i]$ contending ones collide with one another, i.e. $P_c^F(k \mid \omega.X_{m+1}[i])$. Hence,

$$gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_c^F(k \mid \omega.X_{m+1}[i]) \quad (25)$$

4) In the presence of $\omega.X_{m+1}[i]$ contending nodes, k of which collide, the overall energy consumed by the joining nodes is equal to $E_c(k \mid \omega.X_{m+1}[i])$. Thus, we have

$$gen.e_{X_m Y_{m+1}} = \omega.e_{X_m Y_{m+1}} + E_c(k \mid \omega.X_{m+1}[i]) \quad (26)$$

5) Finally, being a possible transition state generated at slot s_i, gen is added to the set Ω_i, that is

$$\Omega_i = \Omega_i \cup gen \quad (27)$$

Epilogue. As it can be observed, when all the N slots in the superframe have been examined, the set Ω_N contains all possible states X_{m+1} reachable from state X_m, in the presence of a slot utilization pattern S_m. Thus, $generate\_states()$ assigns Ω_N to the set $G_{X_m S_m}$, and returns it (line 41). Note that it is possible that two different $transitions$ in $G_{X_m S_m}$ contains the same X_{m+1}. This is because, in general, state X_{m+1} can be reached from state X_m in different ways, depending on the particular events occurred during superframe T_m.

Derivation of the DTMC

Algorithm 7 details the function *generate_DTMC*(N_j, N_A) that takes two parameters as input, namely N_j and N_A, the number of joining nodes and the number of already acquired slots, respectively (line 1). The function returns a

```
1: $(P, E)$ generate_DTMC $(N_j, N_A)$
2:  $Unprocessed = \emptyset$
3:  $Examined = \emptyset$
4:  $F = \emptyset$
5:  $X_j = [N_j,\ 0,\ ...,\ 0]$
6:  $Unprocessed = Unprocessed \cup X_j$
7:  while( $Unprocessed! = \emptyset$ )
8:    extract $X_m$ from $Unprocessed$
9:    $Examined = Examined \cup X_m$
10:   $n_j = \sum_{i=1}^{N} X_m[i]$
11:   for each $S_m : |i : S_m[i] = A| = N_A + N_F - n_j$
12:     $G_{X_m S_m} = generate\_states(X_m, S_m)$
13:     for each $g \in G_{X_m S_m}$
14:       if $g.X_{m+1} \notin Unprocessed \wedge g.X_{m+1} \notin Examined$
15:         $Unprocessed = Unprocessed \cup g.X_{m+1}$
16:       end if
17:       $g.p_{X_m X_{m+1}} = g.p_{X_m X_{m+1}} \cdot \frac{1}{\binom{N}{N_A+N_F-n_j}}$
18:       $F.insert(\{X_m, g.X_{m+1}, g.p_{X_m X_{m+1}}, g.e_{X_m X_{m+1}}\})$
19:     end for
20:   end for
21: end while
22: $P = [\ ]$
23: $E = [\ ]$
24: for each $X_m \in Examined$
25:   for each $f$ in $F : f.X_m = X_m$
26:     $P_{X_m X_{m+1}} += f.p_{X_m X_{m+1}}$
27:   end for
28: end for
29: for each $X_m \in Examined$
30:   for each $f$ in $F : f.X_m = X_m$
31:     $E_{X_m X_{m+1}} += f.e_{X_m X_{m+1}} \cdot \frac{f.p_{X_m X_{m+1}}}{P_{X_m X_{m+1}}}$
32:   end for
33: end for
34: return $(P, E)$
35: end
```

Algorithm 7: Function *generate_DTMC()*

pair of matrices (P, E). In particular, P is the transition probability matrix, and each element P_{XY} indicates the probability that the system state changes from X to Y. Instead, matrix E indicates the energy consumption of joining nodes. Specifically, each element E_{XY} represents the average energy consumed by joining nodes when the network state changes from X to Y.

The *generate_DTMC()* function mainly relies on the following three sets.

• $Unprocessed$, a set including all the states that are still to be examined.

• $Examined$, the set including all network states that have been already examined.

• F: a list of elements $f = \{X_m,\ X_{m+1},\ p_{X_m X_{m+1}},\ e_{X_m X_{m+1}}\}$, each one of which represents one of the possible transitions from a given state X_m to state X_{m+1}. In particular, f is composed of four elements. X_m and X_{m+1} are the initial and final state of transition f, respectively. Instead, $p_{X_m X_{m+1}}$ and $e_{X_m X_{m+1}}$ are the occurrence probability of transition f, and the energy spent by joining nodes during f, respectively.

Initially, the above mentioned sets are created and initialized (lines 2-4). Also, vector $X_j = [N_j,\ 0, ...,\ 0]$, representing the network state at superframe T_j, is created and added to set $Unprocessed$ (lines 5-6). The join procedure terminates when there are no joining nodes that have still not acquired a slot, i.e. the network state is $X_f = [0,\ 0, ...,\ 0]$.

Then, the function performs a *while* loop (lines 7-21), that ends when there are no more states to be analyzed, i.e. when *Unprocessed* $= \emptyset$. At each step of the loop, one element X_m is extracted from *Unprocessed* (line 8), and added to *Examined* (line 9). Thereafter, *generate_DTMC()* generates all possible states X_{m+1} where the network may evolve to from state X_m at a given superframe T_m. The goal is to derive, for every pair $\{X_m, X_{m+1}\}$, both the probability $P_{X_m X_m+1}$ that the system state changes from X_m to X_{m+1}, and $E_{X_m X_m+1}$, i.e. the average energy consumed by joining nodes when such a transition occurs. In order to do that, all the possible slot utilization patterns $S_m = [\bar{s}_1, \bar{s}_2, ..., \bar{s}_N]$ at superframe T_m are considered (line 11). Since i) the network state at superfame T_m is X_m; ii) there were exactly N_A acquired slots when the join procedure started; and iii) we have supposed that the number of joining nodes $N_j = N_F$, then the number of acquired slots at a given superframe T_m is equal to $N_A + N_F - n_j$, where $n_j = \sum_{i=1}^{N} X_m[i]$. Specifically, such a formula calculates the total number of acquired slots at the beginning of T_m, by subtracting the number of joining nodes that are still competing to acquire a slot, i.e. $n_j = \sum_{i=1}^{N} X_m[i]$, from the total number of slots in the superframe, i.e. $N = N_A + N_F$. Having said that, a slot utilization pattern S_m is considered if and only if it satisfies the following condition: $S_m : |\{i \in [1,\ ...,\ N] : S_m[i] = A\}| = N_A + N_F - n_j$, where $|\{\cdot\}|$ indicates the cardinality of a given set. That is, considered slot utilization patterns are such that the number of acquired slots is equal to $N_A + N_F - n_j$. So doing, the function *generate_states()* is invoked for each considered pair $\{X_m, S_m\}$ (line 12). Such a function returns $G_{X_m S_m}$, i.e. the set of all possible *transitions* that can occur given the network state X_m and the slot utilization pattern S_m. Then, for each *transition* $g \in G_{X_m S_m}$, the following operations are performed. First, if state $g.X_{m+1}$ is neither in the set *Unprocessed* nor in the set *Examined*, it is added to *Unprocessed*, since it still has to be analyzed (lines 14-16). Second, the probability $g.p_{X_m X_{m+1}}$, i.e. the probability that transition g occurs, is multiplied by $P(S_m \mid X_m)$, i.e. the probability that the slot utilization pattern is S_m at the beginning of T_m, given the network state X_m (line 17). This is because the probability $g.p_{X_m X_{m+1}}$ is computed considering a specific slot utilization pattern S_m at superframe T_m. Specifically, the probability $P(S_m \mid X_m)$

is equal to $1/\binom{N}{N_A+N_F-n_j}$ where $n_j = \sum_{i=1}^{N} X_m[i]$. In fact, the number of possible slot utilization patterns S_m at superframe T_m is equal to $\binom{N}{N_A+N_F-n_j}$ and all slot utilization patterns S_m are equiprobable. Finally, the tuple $\{X_m,\ g.X_{m+1},\ g.p_{X_m X_{m+1}},\ g.e_{X_m X_{m+1}}\}$ is added to list F (line 18). Once the analysis has been completed, set *Examined* contains all possible system states X_m. Also, list F contains all possible transitions from a given state X_m to any other state X_{m+1}. Then, *generate_DTMC()* computes matrix P (lines 24-28). Specifically, the probability to reach state $Y \in Examined$ from state $X \in Examined$ is computed by considering all elements f in F such that $f.X_m = X \wedge f.X_{m+1} = Y$. More specifically, the probability to reach state XY from state X, i.e. P_{XY}, is calculated as:

$$P_{XY} = \sum_{f: f.X_m = X \wedge f.X_{m+1} = Y} f.p_{X_m X_{m+1}} \tag{28}$$

Finally, the matrix E is computed (lines 29-33). Specifically, the average energy spent by joining nodes when the network state changes from X to Y, i.e. E_{XY}, is calculated as:

$$E_{XY} = \sum_{f: f.X_m = X \wedge f.X_{m+1} = X} f.e_{X_m X_{m+1}} \cdot \frac{f.p_{X_m X_{m+1}}}{P_{XY}} \tag{29}$$

That is, E_{XY} is calculated as the weighted sum of energies $f.e_{X_m X_{m+1}}$ spent by joining nodes during each possible transition f. As weight for each f, we consider $\frac{f.p_{X_m X_{m+1}}}{P_{XY}}$, i.e. the probability that the specific transition f occurs, given an occured transition from X to Y.

E. Analysis of the Centralized Solution

In Section 8 we derived formulas to compute the overhead due to the JAMMY join procedure. Let us now consider the overhead of the join process when the centralized solution is used. We assume that each joining node transmits an association message to the Coordinator node, in order to join the network. We further assume that all association messages are successfully transmitted to the Coordinator node during superframe T_j, i.e., the joining process is completed in one single superframe. It follows that, starting from superfame T_{j+1}, all joining nodes receive the slot utilization pattern by the Coordinator node and, hence, are fully operative. We want to point out that this is an optimistic assumption since joining nodes have typically to compete with each other to access the medium and transmit their association messages. Hence, the joining process may require more than one superframe to be completed.

The energy E_k^C consumed by the joining nodes during superframe $T_{j+k},\ k \geq 0$, when the centralized solution is used, can be calculated according to the following equation.

$$E_k^C = \begin{cases} N_j \cdot (P_{TX} D_{ass}) & k = 0 \\ N_j \cdot (P_{TX} D_{tx} + P_{RX} D_{ack} + P_{RX} D_{sup}) & k > 0 \end{cases} \tag{30}$$

where D_{ass} and D_{sup} are the time required to transmit the association message and the slot utilization pattern, respectively.

Equation 30 can be explained as follows. At superframe T_j ($k = 0$), the energy consumption is equal to $N_j \cdot (P_{TX} \cdot D_{ass})$ since the N_j joining nodes transmit one association message each. Conversely, from superframe T_{j+1}, every joining node 1) receives the slot utilization pattern from the Coordinator node, and 2) transmits its data packet and receives the related acknowledgment. Hence, the total energy consumed by the N_j joining nodes during superframe $T_{j+k},\ k > 0$, is equal to $N_j \cdot (P_{TX} D_{tx} + P_{RX} D_{ack}) + N_j \cdot (P_{RX} D_{sup})$ where the first term is the energy due to the data packet transmissions while the second term accounts for the reception of the slot utilization pattern.