DoS detection in WSNs: Energy-efficient methods for selecting monitoring nodes

Quentin MONNET[1], Lynda MOKDAD[1*], Paolo BALLARINI[2], Youcef HAMMAL[3], Jalel BEN-OTHMAN[4]

[1] *Université Paris-Est, LACL (EA 4219), UPEC, France*
[2] *Université Paris-Saclay, Centrale-Supélec, MICS, France*
[3] *University of STHB, LSI Laboratory, Algeria*
[4] *Université Paris 13, L2TI (EA 3043), France*

SUMMARY

The use of wireless sensor networks (WSNs) has increased rapidly over the last years. Due to their low resources, sensors come along with new issues regarding network security and energy consumption. Focusing on the network availability, previous studies proposed to protect clustered network against denial of service attacks with the use of traffic monitoring agents on some nodes. Those control nodes have to analyze the traffic inside a cluster and to send warnings to the cluster-head whenever an abnormal behavior (*e.g.,* high packets throughput, or non-retransmission of packets) is detected. But if the control nodes (*cNodes*) die out of exhaustion, they leave the network unprotected. To better fight against attacks, we try to enhance this solution by renewing periodically the election process. Furthermore, we propose three energy-aware and secure methods to designate the *cNodes* in a hierarchically clustered WSN. The first one is a simple self-election process where nodes randomly designate themselves. It leads to a better load balancing than a static method (*i.e.*, with no renewal), but we argue that we can obtain better results by considering the remaining energy of the nodes at *cNodes* selection time. Hence the second algorithm is purely based on the residual energy of the sensors. We discuss limitations of this deterministic process concerning security and cluster coverage, and suggest workarounds. These improvements lead us to the third mechanism. It is based on residual energy too, but it includes a democratic election process in which nodes in the cluster vote to optimize the *cNode* role attribution. Results obtained from simulation experiments with the `ns-2` tool are provided to analyze the energy repartition in the network and to compare the three selection algorithms. All experimental outcomes show improvements of the load balancing in the network, while maintaining good detection coverage, in regard to static selection. Furthermore, the analysis of the respective performances of the three mechanisms is used as a basis to establish recommendations regarding the use cases of those methods.

KEY WORDS: Wireless sensor networks; Reliability, availability, and serviceability; Energy-aware systems; Simulation.

Published on 14th September 2017
in
Concurrency and Computation – Practice and Experience
Volume 29, Issue 23
Combined Special issues on Applications and techniques in information and network security (CSTA2015) and International conference on innovative network systems and applications held under the federated conference on computer science and information systems (FedCSis-INetSApp2015)
https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4266

*Correspondence to: Prof. Lynda Mokdad,
Université Paris-Est Créteil, LACL, 61 avenue du Général de Gaulle, 94010 Créteil Cedex, France
E-mail: lynda.mokdad@u-pec.fr

1. INTRODUCTION

Wireless sensor networks *Smart cities* or the *Internet of Things* are foreseen to deeply change people's daily lives. Such projects will interconnect a multitude of devices and bring many functionalities to the end users through an extensive use of sensors. Ambient light, temperature, air pollution degree measurement, or traffic monitoring are just a few examples of applications involving those sensors. There will be sensors everywhere, to gather amounts of data that human beings alone could not measure: sensors deployed as networks can perform constant monitoring tasks over wide—and sometimes hard to access—areas.

Such networks are called *wireless sensor networks* (WSNs). The sensors (or *nodes*) are small devices able to gather data on their physical environment. They communicate with one another through radio transmission, but they have low resources at their disposal: limited computing power, limited memory, as well as a limited battery [1, 2]. They are often dropped into hostile areas (by helicopter for instance), or may generally be difficult to access, so the batteries must be considered as single-use. The sensors have to self-organize themselves and to deploy low-consuming routing algorithms so as to create a functional network. All relevant data is typically forwarded to an entity called the *base station* (BS), which does not have the same limitations as the sensors, and acts as an interface between the WSN and the operator (or the external world) as displayed on Figure 1.

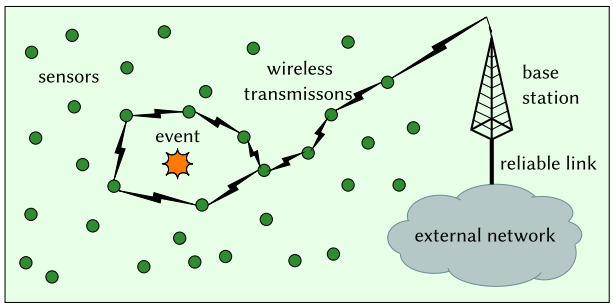

Figure 1. Clustered wireless sensor networks scheme

Wireless sensor networks may be deployed for all kinds of applications, some of them being crucial. For instance there is a lot at stakes when sensor networks are used for forest fires detection. Critical cases also involve all military uses of the sensors: they can be used to detect the presence of biological, chemical or nuclear agent, or to monitor infantry units over battlefields [3]. Such contexts bring strong requirements in terms of security guarantees for the network. Various works deal with ways of preventing unauthorized access to data, or with the necessary precautions to guarantee data authenticity and integrity inside WSNs [4, 5]. But confidentiality as well as authentication are of little use if the network is not even able to deliver its data correctly.

Denial of Service in WSNs Denial of Service (DoS) attacks indeed aim at reducing, or even annihilating, the network ability to achieve its ordinary tasks, or at preventing a legitimate agent from using a service [6]. Because of the limited resources of their nodes, WSNs tend to be rather vulnerable to DoS attacks. Concrete attacks include jamming the communications, monopolizing the channel ("greedy" attacks) or attempting sleep deprivation on "normal" sensors, for example. They are launched from the outside as well as from the inside of the network: a compromised sensor node can be used in order to send corrupted data at a high rate, either to twist the results or to drain the nodes' energy faster. Attacks can target all layers of the network, although we mainly focus here on the Media Access Control (MAC) and routing layers. The problem we tackle is the development and analysis of detection mechanisms that are efficient both in terms of detection (*i.e.*, they guarantee a high rate of detection

of compromised nodes) and in terms of energy (*i.e.*, they guarantee a balanced energy consumption throughout the network).

Clustered WSNs One way to save some battery power during communications may reside in the choice of the network architecture and of the protocol used to route data from a sensor to the BS. In a hierarchical WSN, the network is divided into several clusters. The partition is done according to a clustering algorithm such as LEACH [7, 8], HEED [9], or one based on ultra-metric properties [10, 11]. In each cluster, a single common node is designated and becomes a cluster head (CH), responsible for directly collecting data from the other nodes in the cluster. Once enough data has been gathered, the CHs proceed to data aggregation [12]. Then they forward their data to the BS. CHs are the only nodes to communicate with the BS, either directly, through a long-range radio transmission, or by multi-hopping through other CHs (see Figure 1). So as to preserve the nodes' energy as long as possible, the network reclustering is repeated periodically, with different nodes being elected as CHs. Note that clustering is not limited to a "single-level" partition. We can also subdivide a cluster into several "subclusters". The CHs from those "subclusters" would then send their aggregated data to the CHs of their parent clusters.

DoS detection: from static to dynamic guarding policies In a hierarchically organised WSN, a control node (*cNode* in the remainder of this article) is a node that is chosen to analyze the traffic directed to the CH of the cluster it belongs to, and potentially detect any abnormal behavior. Therefore, *cNodes* provide us with an efficient way to detect DoS attacks occurring in the network. Note that *cNodes* are only meant to detect DoS attacks, thus they do not perform any sensing, nor do they send any data (apart from attack detection alarms). *cNodes*-based detection was first presented in [13], but the authors do not mention any periodical (*cNodes*) re-election scheme. One can suppose that the renewal of the election occurs each time the clustering algorithm is repeated. In [14], we proposed a dynamic approach: *cNodes* are re-elected periodically (any node in a cluster may be chosen, except the CH) with the election period selected to be shorter than that between two network clusterings. Intuitively such a dynamic approach (in comparison to that of [13]), leads to more uniform energy consumption while preserving good detection ability.

Our contribution We propose a dynamic renewal of the designation process of the *cNodes*. The process itself can be performed by applying different algorithms:

1. Nodes can selected in a random fashion, but it is possible to achieve a better balance of the energy consumption.
2. Thus we also propose a second designation process which is based on energy, in order to obtain better performances. We propose to designate the sensors for the *cNode* position according to their residual energy, but we show that several problems occur with deterministic election. Indeed, compromised nodes could see a flaw to exploit in order to take over the *cNode* role and decrease the odds of being detected by announcing high residual energy. We address this issue by introducing a second role of surveillance: we choose "*vNodes*" responsible for watching over the *cNodes* and for matching their announced consumption against mathematical model. We also recommend that every node in the cluster is monitored by at least one *cNode*, to prevent all the *cNodes* to be elected inside the same spatial area of the cluster at each election iteration.
3. This second method leads to a better equilibrium of the energy flow inside the cluster, but the overall consumption increases because of the *vNodes*. This is why we propose a third method called "democratic election" process, which reuses the observations performed by the *cNodes* to eliminate the need for *vNodes*. Furthermore, this third election method can use additional parameters such as index connectivity or signal power for selecting the optimal monitoring nodes.

The three selection methods are evaluated with the `ns-2` tool. We analyze the numerical results and provide comparison of the three processes, as well as recommendations on use case scenarios for each method.

Outline The remainder of this work is organized as follows: Section 2 presents related work. In Section 3 we give an overview of DoS attack detection for cluster-based WSNs, based on the use of monitoring nodes. It is followed by three sections introducing three distinct selection processes for these monitoring nodes. Thus Section 4 presents the random-based selection mode, while Section 5 introduces a scheme based on residual energy of the nodes, tackling the issues of a deterministic algorithm. Section 6 deals with the self-election process used to improve performances upon the method based on residual energy. Numerical results obtained through network simulations with the `ns-2` tool are provided in Section 7, and they lead to comparison of the three selection methods and to usage recommendations in Section 8. Finally, Section 9 permits us to sum up our contributions and to consider future work leads.

2. RELATED WORK

This section is divided into three parts: security in wireless sensor networks, denial of service specific mechanisms, and clustering algorithms and energy preservation.

2.1. Security in WSNs

Denial of service is not the only type of attack a WSN should resist to. Security in general in sensor networks has attracted quite a lot of interest during the last few years. Hence it has been the subject of many studies in literature, as well as several state-of-the-art articles [15, 16].

Confidentiality and integrity must be ensured to prevent attackers access to or tampering with sensitive data. A number of solutions have been proposed [8], many of them involving strong [4] and/or homomorphic [17] cryptography, some relying on other mechanisms such as multi-path based fragmentation of the packets [18] or game theory [19].

Authentication brings to participants the guarantee that the peer they are communicating with truly is what it pretends to be; that is another important point. It has been deeply investigated as well [20]. Many lightweight proposals for key management in WSNs have been suggested [21, 22].

Apart from those, there have been a variety of proposals to secure other elements, on a basis than any information about any aspect of the network might be valuable to an attacker. Hence there are approaches, for instance, to secure the geographical location of the nodes through epidemical information dissemination [23] as well as through more conventional mechanisms [24].

2.2. DoS-specific mechanisms

Denial-of-service attacks embrace many different attacks, which can target all layers of the network [25]. Jamming the radio frequencies as well as disturbing the routing protocols are just two examples of ways to harm the network. In reaction to these, a number of solutions have been proposed [26]. As stated in the introduction, we focus in this paper on inside attackers attempting to bend the MAC protocol parameters to their needs, be it to achieve better performances for themselves (greedy attacks) or to generally harm the network (jamming attacks or sleep deprivation). To detect such attackers, many solutions rely on trust models [27, 28] with agents applying a set of rules [29] on traffic to attribute a trust value to each of the nodes in the network. Below are outlined some notable proposals.

Back in 2001, most work focused on making WSNs feasible and useful. But some people already involved themselves into security. For instance, SPINS (Security Protocols for Sensor Networks) was proposed in [30] to provide networks with two symmetric key-based security building blocks. The first block, called SNEP (Secure Network Encryption Protocol), provides data confidentiality, two-party data authentication and data freshness. The second block, called μTESLA ("micro" version of the Timed, Efficient, Streaming, Loss-tolerant Authentication Protocol) assumes authenticated broadcast using one-way key chains constructed with secure hash functions. No mechanism was put forward to detect DoS attacks.

The best way to detect for sure a DoS attack in a WSN is simply to run a detection mechanism on each single sensor. Of course, this solution is not feasible in a network with constraints. Instead of fitting out each sensor with such mechanism, it is proposed in [31] to resort to heuristics in order to set a few nodes equipped with detection systems at critical spots in the network topology. This optimized placement enables distributed detection of DoS attacks as well as reducing costs and processing overheads, since the number of required detectors is minimized. But those few selected nodes are likely to run out of battery power much faster than normal nodes.

Some works examine the possibility of detecting the compromising of nodes as soon as an opponent physically withdraws them from the network. In the method that is developed in [32], each node keeps a watch on the presence of its neighbors. The Sequential Probability Radio Test (SPRT) is used to determine a dynamic time threshold. When a node appears to be missing for a period longer than this threshold, it is considered to be dead or captured by an attacker. If this node is later redeployed in the network, it will immediately be considered as compromised without having a chance to be harmful. Nothing is done, however, if an attacker manages to compromise the node without extracting the sensor from its environment.

In [33], a revised version of the OLSR protocol is proposed. This routing protocol called DLSR aims at detecting distributed denial of service (DDoS) attacks and at dropping malicious requests before they can saturate a server's capacity to answer. To that end, the authors introduce two alert thresholds regarding this server's service capacity. The authors also use Learning Automata (LAs), automatic systems whose choice of next action depends on the result of its previous action. There is no indication in their work about the overhead or the energy load resulting from the use of the DLSR protocol.

A novel broadcast authentication mechanism can also be deployed so as to cope with DoS attacks in sensor networks such as in [34]. This scheme uses an asymmetric distribution of keys between sensor nodes and the BS, and uses the Bloom filter as an authenticator, which efficiently compresses multiple authentication information. In this model, the BS or sink shares symmetric keys with each sensor node, and proves its knowledge of the information through multiple MAC values in its flooding messages. When the sink floods the network with control messages it constructs a Bloom filter as an authenticator for the message. When a sensor node receives a flooded control message, it generates their Bloom filter with its keys and in the same way the sink verifies message authentication.

Much of our work relies on the work of Lai and Chen who proposed in [13] a system detection based on static election of a set of nodes called "guarding nodes" which analyze traffic in a clustered network. When detecting abnormal traffic from a given node, "guarding nodes"—we call them *cNodes*—identify it as a compromised node and inform the cluster head of it. On reception of reports from several distinct *cNodes* (to prevent false denunciation from a compromised node), the CH virtually excludes the suspicious node from the cluster. The authors show the benefit of their method by presenting numerical analysis of detection rate. Although the method is efficient for detecting rogue nodes, the authors do not give details of the election mechanism for choosing the *cNodes*. Also, there is no mention in their study of renewing the election in time, which causes the appointed *cNodes* to endorse heavier energy consumption on a long period.

2.3. Clustering algorithms and energy preservation

A lot of approaches intended to bring security into a WSN are cluster-based [35]. But the main purpose of clustering a sensor network usually resides in scaling possibilities, improved nodes management and energy savings brought by partitioning. Several clustering algorithms have been proposed [36]. They generally aim at determining which nodes in the network will be the cluster heads, often basing the choice on energetic considerations. Basically, choosing a cluster head in a network is not so different than selecting *cNodes* in a cluster. But in the latter case we have some additional constraints on security.

One of the easiest clustering algorithms to implement, and probably one of the most used, is the LEACH algorithm [37].

2.3.1. LEACH functioning LEACH is likely one of the easiest algorithm to apply to recluster the network. It is a dynamical clustering and routing algorithm. It splits a set of nodes into several subsets, each containing a cluster head. This CH is the only node to assume the cost-expensive transmissions to the BS.

Here is the LEACH detailed processing. Let P be the average percentage of clusters we want to get from our network at an instant t. LEACH is composed of cycles made of $\frac{1}{P}$ rounds. Each round r is organized as follows:

1. Each node i computes the threshold $T(i)$:

$$T(i) = \begin{cases} \dfrac{P}{1 - P \cdot \left(r \bmod \frac{1}{P}\right)} & \text{if } i \text{ has not been CH yet} \\ 0 & \text{if } i \text{ has already been CH} \end{cases}$$

   Each node chooses a pseudo-random number $0 \leq x_i \leq 1$. If $x_i \leq T(i)$ then i designates itself as a CH for the current round. $T(i)$ is computed in such a way that every node becomes CH once in every cycle of $\frac{1}{P}$ rounds: we have $T(i) = 1$ when $r = \frac{1}{P} - 1$.
2. The self-designed CHs inform the other nodes by broadcasting a message with the same transmitting power, using carrier sense multiple access (CSMA) MAC.
3. The other nodes choose to join the cluster associated to the CH whose signal they receive with most power. They message back the CH to inform it (with the CSMA MAC protocol again).
4. CHs compile a "transmission order" (time division multiple access, TDMA) for the nodes which joined their clusters. They inform each node at what time it is expected to send data to its CH.
5. CHs keeps listening for the results. Normal sensors get measures from their environment and send their data. When it is not their turn to send, they stay in sleep mode to save energy. Collisions between the transmissions of the nodes from different clusters are limited thanks to the use of code division multiple access (CDMA) protocol.
6. CHs aggregate, and possibly compress the gathered data and send it to the BS in a single transmission. This transmission may be direct, or multi-hopped if relayed by other CHs.
7. Steps 5 and 6 are repeated until the round ends.

It is possible to extend LEACH by adding the remaining energy of the nodes as a supplementary parameter for the computation of the $T(i)$ threshold.

Note that each node decides whether to self-designate itself as a CH or not. Its decision does not take into account the behavior of surrounding nodes. For this reason, we can possibly have, for a given round, a number of CHs very different from the selected percentage P. Also, all the elected CHs may be located in the same region of the network, leaving "uncovered" areas. In that case, one can only hope that the spatial repartition will be better during the next round.

2.3.2. LEACH improvements There are a number of proposals derived from LEACH, to improve either its efficiency [38, 39] or its security. In [40], the authors propose to add security mechanisms via a revised version of LEACH protocol. SecLEACH uses random key predistribution as well as μTESLA (authenticated broadcast) so as to protect communications. But the authors do not mention any mechanism to fight DoS attacks.

In [41], the authors propose another way to secure the LEACH protocol against selfish behaviors, using elements from game theory. With S-LEACH, the BS uses a global Intrusion Detection System (IDS) while LEACH CHs implement local IDSs. The interactions between nodes are modeled as a Bayesian game, that is, a game in which at least one player (here, the BS) has incomplete information about the other player(s) (in this case: whether the sensors have been compromised or not). Each node has a "reputation" score. Selfish nodes can cooperate (so as to avoid detection) or drop packets. The authors show that this game has two Bayesian Nash equilibriums which provide a way to detect selfish nodes, or to force them to cooperate to avoid detection.

Other algorithms Other possible clustering algorithms include HEED [9], which is designed to save more energy than standard LEACH, and could lead to a better spatial repartition of the CHs inside the network. But in our network, all the sensors have the same initial available energy, and every one of them is able to directly reach the BS if need be. Under those assumptions, LEACH may not consume more energy than HEED protocol, and remains easier to use.

Note that, aside from clustering, the importance of energy issues in WSNs has led to proposals of several mechanisms to cut down its consumption [42], based for example on packet priority [43].

3. DESCRIPTION OF DOS DETECTION METHOD

3.1. Wireless Sensor Networks

We focus on the problem of detecting denial of service (DoS) attacks in a WSN [44]. We recall that a WSN consists of a finite set of sensors plus a fixed base station (BS). Traffic in a WSN (mainly) flows from sensor nodes towards the BS. Furthermore since WSN nodes have inherently little energy, memory and computing capabilities, energy efficiency is paramount when it comes with mechanisms/protocols for WSN management. In the following we assume that the mobility of the nodes is limited or null.

Our goal is to set an efficient method to detect compromised nodes which may try to corrupt data, or to saturate the network's capacity, by sending more data than it should. In this case, efficiency can be measured in two ways:

- the detection rate of the compromised node(s);
- the network's lifetime, as we want to spend as little energy as possible.

In order to achieve these goals we focus on the following techniques: hierarchical network clustering, and dynamical election of control nodes responsible for monitoring the traffic.

3.2. Hierarchical clustering

The class of WSNs we consider is that of hierarchically cluster-based networks. The set of sensors has been partitioned into several subsets, which are themselves split into subclusters. For more clarity, we will call 1-clusters the sets resulting from the first clustering of the global set, and k-clusters the subset issued from the splitting of any $(k-1)$-cluster. The successive clusterings are carried out with the use of any existing clustering algorithm, such as LEACH [7, 8], HEED [9], algorithms based on ultra-metric properties [12], *et cætera*. Each cluster contains a single cluster head (CH), designated among the normal nodes. The CH is responsible for collecting data from the other nodes of the subset. To follow up our naming conventions, we will call k-CHs the CHs belonging to the k-clusters. The k-CHs send the data they gathered to their $(k-1)$-CH, the "0-CH" being the base station. In that way, the k-CHs are the only nodes to send packets to the $(k-1)$-CHs. Normal nodes' transmissions do not have to reach the base station directly, which would often consume much more energy than communicating with a neighbor node. An example 2-clustered network is displayed on Figure 2.

k-LEACH Once a clustering algorithm has been applied to the network to determine a first set of clusters, nothing prevents us to apply it again on each cluster. This is the way we got our k-clusters: we applied the LEACH algorithm k times recursively. We call those recursive iterations the k-LEACH algorithm. In practice, we had k equal to 2, for the following reasons:

- so as to save more energy than what we would do with 1-LEACH;
- so as to have a finer clustering of the network, in order to elect control nodes in each of the 2-clusters, to maximized the cover area and the probability of detecting compromised nodes.

3.3. Attacks detection through cNodes

Along with normal nodes and cluster heads, a third type of node is present in the lower k-clusters of the hierarchy. The *cNodes*—for control nodes—were introduced in [13] to analyze the network traffic and to detect any abnormal behavior from other nodes in the cluster.

Control nodes watching over the input traffic allow the detection of various types of denial of service attacks. This is achieved with agents running on the *cNodes* and applying specific rules on overheard traffic [29]. Each rule is used to fight against one (sometimes a few) specific attack(s): jamming, tampering, black hole attacks, and so on. Each time a *cNode* notices that a rule is broken by a node, it raises a bad behavior for this node, and send an alert to the cluster head. Following are some example rules:

- Rate rule: assuming that minimal and maximal rates for data each node sends are enforced, a bad behavior will be reported if those rates are not respected. With this rule, monitoring agents should be able to detect negligence (if minimal rate is not reached) or flooding (if maximal rate is exceeded) attacks.
- Retransmission rule: a *cNode* overhearing a packet supposed to be retransmitted by one of its neighbor (the neighbor node is not the final destination for this packet). If the concerned neighbor does not forward the packet, it may be undertaking a black-hole (full dismissal of packets) or a selective forwarding attack.
- Integrity rule: a bad behavior will be raised if a neighbor of the node running the monitoring agent tampers with a packet before forwarding it. Applying this rule assumes that the nodes are not expected to proceed either to data aggregation or compression before forwarding.
- Delay rule: forwarding a packet should not exceed a threshold delay.
- Replay rule: a message should be sent no more than a limited number of times.
- Jamming rule: an unusually high number of collisions (compared to average, or concerning only some nodes) may be related to the presence of a jamming node. If jamming is done with random noise, without legitimate packets containing a node identifier, it may be difficult to detect the source of it, but several cooperating agents should be able to detect it.
- Radio transmission range rule: a node sending messages with an unexpectedly high power may be trying to launch a hello flood (it tries to appear in the neighbor list of as many nodes as possible) or

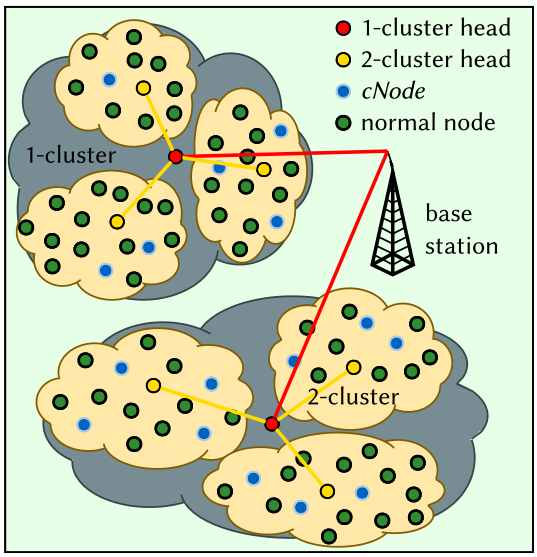

Figure 2. Scheme of a twice clustered WSN

wormhole attack (it redirects a part of the overheard traffic to another part of the network). Hence it may be consider as a bad behavior.

In the rest of this study, we will not describe in details each one of the mentioned attacks, nor will we detail the associated solutions to counter them. When details are needed, we will consider only one example: flooding attacks. The model of a flooding attack is the following: a malicious node sends a high amount of data to prevent legitimate nodes from communicating by saturating the medium, or by establishing too many connections with the receiver node [45]. In wireless sensor networks, it is also used to drain the energy of neighbor nodes.

So *cNodes* analyze the input traffic for the 2-CH of their 2-cluster, and watch out for abnormal traffic flows. Detection takes place whenever a rule is broken. In that case the *cNode* sends a warning message to the CH. In order to prevent a compromised *cNode* to declare legitimate nodes as compromised the detection protocol requires that the CH receives warnings by a minimum number of distinct *cNodes* before actually recognising the signaling as an actual anomaly. Once the CH has received warnings from a sufficiently large number of distinct *cNodes* it starts ignoring the packets coming from the detected compromised sensor. *cNodes* may also monitor output traffic of the CHs and warn the BS if they come to detect a compromised CH.

Observations *cNodes* apply a very basic trust based scheme to the cluster: when a sensor node breaks a rule, for example by exceeding a given threshold for transmitted packets, it is considered as untrustworthy. There are many other trust based schemes in the literature, most of them more advanced than this one (see Section 2). The *cNodes* could implement several other trust mechanisms (by lowering a score on bad behavior for each node, for instance). As more complex mechanism would create additional overhead, we prefer to limit ourselves to this simple method in this study.

3.4. Periodical renewing of the cNodes

cNodes are periodically elected among normal sensors. The guarding functionality of *cNodes* may lead to an energy consumption higher than that of "normal" (*i.e.*, sensing) nodes. In order to maximize the repartition of the energy load, we propose a scheme by which a new set of *cNodes* is periodically established with an selection period shorter than the length of a LEACH round (that is, the period between two consecutive CH elections).

This dynamical renewing of the selection process is an essential part of our proposal. Many of the recent intrusion detection systems proposed for WSNs tend to be lightweight, to consume little energy. We believe that a dynamical renewing of the selected *cNodes* helps a lot to balance the load inside the cluster. Depending on the application running in the network, maybe this balancing is not worth the constraints induced by periodical re-election, but generally energy preservation is a priority in WSNs and distributing the consumption among all the nodes helps to maintain the highest possible amount of nodes in activity for as long as possible. Also, lightweight IDSs themselves may be designed to minimize the disparities in energy consumption inside a network, but we argue that with a system as simple as the *cNodes*, simulation results indicate that the savings are not negligible.

A second thing to consider is that the recursive clustering as well as the dynamic renewing of the monitoring nodes can be used with other detection systems than the *cNodes* we use here. If an IDS is good at preserving energy and balancing nodes, but needs to be run only by a subset of the nodes in the network, dynamical selection processes presented in this work can be applied so as to select the sensors which will run the system (provided the monitoring sensors do not need any specific hardware that would differentiate them from the "normal" nodes).

Renewing the *cNodes* implies running a selection algorithm. There are many ways to proceed: in following sections, we present three different methods permitting to designate the *cNodes* for a new period, with their respective pros and cons.

4. RANDOM SELECTION

The first method for *cNodes* selection is a random-based selection: a pseudo-random number generator is used to determine the set of *cNodes*.

We propose three possible implementations for this process: self-election as for the CHs, election processed by the CHs and election processed by the BS.

Distributed self-election The first possibility to process to the random selection of the *cNodes* is to reuse the distributed self-designation algorithm defined for the election of the CHs. With this implementation, each non-CH node chooses a pseudo-random number comprised between 0 and 1. If this number is lower than the average percentage of *cNodes* in the network that was fixed by the user, then the node designates itself as a *cNode*. Otherwise, it remains a normal sensor.

This method has two drawbacks. First, each node has to compute a pseudo-random number, which may not be necessary with other methods. Second, each node chooses to designate (or not) itself, without taking into account at any moment the behavior of its neighbors. As a result, the election proceeds with no consideration for the clustering that has been realized in the network. Indeed it is unlikely that the set of elected *cNodes* will be uniformly distributed among the 2-clusters that were formed, and it is even possible to end up with some 2-clusters containing no *cNodes* (thus being completely unprotected against attacks).

A possible workaround for this second drawback could be a two-steps election: in a first round nodes self-designate (or not) themselves. Then they signal their state to the 2-CHs they are associated to. In the second round, the 2-CHs may decide to designate some additional *cNodes* if the current number of elected nodes in the cluster is below a minimal percentage.

CH-centralized election The second possibility is to get the *cNodes* elected by the 2-CHs. In this way, each 2-CH elects the required number of *cNodes* (*i.e.*, corresponding to user specifications). For example, if the 2-cluster contains 100 nodes and the desired percentage of *cNodes* in the network is 10 %, the 2-CH will compute 10 pseudo-random numbers and associate them with node IDs corresponding with sensors of its 2-cluster. This solution is computationally less demanding as only the 2-CHs have to run a pseudo-random number generation algorithm. However it has yet another drawback: if a CH gets compromised, it won't be able to elect any *cNode* in its cluster, thus leaving the cluster open to attacks. As with the LEACH protocol, every sensor node becomes, sooner or later, a CH, the problem may occur for any compromised node hence propagating, potentially, throughout the network. Note that, nothing prevents a compromised sensor to declare itself as a CH node to the others at any round of the LEACH algorithm.

This method is the one that we have implemented in our `ns-2` simulation.

BS-centralized election The third method consists in a centralized approach where the BS performs *cNodes* election. With this method CHs send the list of nodes that compose their clusters to the base station and the BS returns the list of elected *cNodes*. Observe that, opposite to sensor nodes, the BS has no limitation in memory, computing capacity or energy. Thus the clear advantage of BS-centralized election is that all costly operations (*i.e.*, pseudo-random numbers calculation) can be reiterated in a (virtually) unconstrained environment (*i.e.*, the BS) This technique is explained in detail in [14].

From a robustness point of view note that this method is not completely safe either. In fact, if a compromised node was to declare itself as a CH, its escape method to avoid detection would consist of declaring its cluster as empty (*i.e.*, by sending an empty list instead of the actual sensors in its cluster to the BS). In this case, the BS would not elect any *cNode* in its cluster, hence the compromised CH would not be detected. To avoid such a situation, the BS should react differently in case it receives an indication of empty cluster from some nodes. Specifically, in this case, the BS would have to consider that nodes not detected as or by CHs might not simply be dead, and thus still consider them as eligible *cNodes*. The

main drawback of this method is that the distributed nature of election (together with its advantages) is completely lost.

The random selection method is easy to deploy, and has little overhead; but it does not balance the energy consumption very well inside the cluster. In order to obtain better performances, we could for instance consider the available residual energy at *cNodes* selection time.

5. SELECTION BASED ON RESIDUAL ENERGY

5.1. Selection process

Electing the *cNodes* is not an easy task. In previous section we exposed and compared three possible implementations for random selection of those sentry nodes:

- pseudo-random election by the base station;
- pseudo-random election by the cluster head;
- pseudo-random election by the nodes themselves.

We assumed that election should be random so that compromised nodes would not be aware of which node could control the traffic. We did not consider the remaining energy during the *cNodes* election. But monitoring the traffic implies to keep listening for wireless transmission without interruption. Hence *cNodes* will have a greater energy consumption than normal nodes. Given that preserving energy is an essential issue in the network, we now prefer to ensure load balancing rather than assuring a pseudo-random election, and thus to consider the residual energy of the nodes during the election. This choice also raises new issues and makes us define a new role for the nodes in the cluster[†].

5.2. Using vNodes to ensure a secured deterministic election

The issue with energy measurement is that no agent in the network is able to measure the residual energy of a given node N, but the node itself. The neighbor nodes of N may record messages sent from N and compute a rough estimate, but as they know neither the initial amount of energy of N (at the network deployment) nor the energy N spent for listening, estimates can not be used to obtain values precise enough so as to reliably sort the nodes according to their residual energy.

So the only way to get the residual energy of a node is to ask this node. The election algorithm we propose is described as follows:

1. During the first step, each node evaluates its residual energy and sends the value to the cluster head;
2. Having received the residual energy of all nodes in the cluster, the cluster head picks the n nodes with the highest residual energy (where n is the desired number of *cNodes* during each cycle) and returns them a message to assign them the role of *cNode*.

It is a deterministic selection algorithm that eliminates any random aspect from the process. The rule is simple: nodes possessing the highest residual energy will be elected. Given that the *cNode* role implies consuming more energy (*cNodes* listen to surrounding communications most of the time), rotation of the roles is theoretically assured. But the deterministic aspect is also a flaw that may be exploited by compromised nodes. This is a crucial issue: we can not neglect compromised nodes as the whole *cNodes* mechanism is deployed in the sole purpose to detect them!

More precisely, the problem may be stated as follows. Compromised nodes will be interested in endorsing a *cNode* role, as it enables them:

[†]The recursive k-clustering is used in this section in the same way as in former section. And yet for simplicity we will only mention “clusters”, as the solution is not dependent of the depth of recursive clustering.

- to reduce the number of legitimate *cNodes* able to detect them;
- to advertise the cluster head about "innocent" sensing nodes to have them revoked.

When a pseudo-random election algorithm is applied, a compromised node (or even several ones) can be elected during a cycle, but it will loose its role further in time, for later cycles. Even with a self-election process, compromised nodes can keep their *cNode* role as long as they want, but they can not prevent other (legitimate) nodes to elect themselves, too. With deterministic election, however, they can monopolize most of the available *cNode* roles. They only have to announce the highest residual energy value at the first step of the election to get assured to win. If there are enough compromised nodes to occupy all of the n available *cNode* roles, then they become virtually immune to potential detection.

To prevent nodes from lying when announcing their residual energy, we propose to assign a new role to some of the neighbors of each *cNode*. Those nodes—we call them *vNodes*, as for *verification* nodes—are responsible for the surveillance of the monitoring nodes. Once the *cNodes* election is over, each neighbor to a *cNode* decides with a given probability whether it will be a *vNode* for this *cNode* or not. A given node can act as a *vNode* for several *cNodes* (in other words, it can survey several neighbor *cNodes*).

If this role consumes too much energy, it is not worth deploying *vNodes*: we should rather use pseudo-random election for the *cNodes*. So *vNodes* must not stay awake and listen most of the time, as *cNodes* do. Instead they send, from time to time, requests to the *cNode* they watch over, asking it for its residual energy. They wait for the answer, and keep the value in memory.

Once they have gathered enough data, *vNodes* try to correlate the theoretical model of consumption of the *cNode* they survey and its announced consumption, deduced from broadcast messages (during elections) and answers to requests from *vNodes*. Four distinct cases may occur:

1. The announced consumption does not correlate (at all) with the theoretical model: there is a high probability the node is compromised and seeks to take over *cNode* role. It is reported to the cluster head.
2. The announced consumption correlates *exactly* with the theoretical model: the node is probably a compromised node trying to get elected while escaping to detection (in other words, the rogue *cNode* adapts its behavior regarding to the previous point). It is easy to detect the subterfuge as values received from the rogue node and the ones computed by the *vNodes* are exactly the same. It is reported to the cluster head.
3. The announced consumption correlates roughly with the theoretical model, but does not evolve in the same way (with regard to the model) as the real consumption locally observed by the *vNodes* (local (in time) evolution of the announced consumption does not "stick" to the one of the surrounding *vNodes*, which should roughly rise or decrease during the same periods). The node is probably compromised, trying to escape detection by decreasing its announced energy with random values. It is reported to the CH.
4. The announced consumption correlates roughly with theoretical model, and evolves in the same way as the traffic observed by *vNodes*. Whether the node is compromised or not, it has normal behavior and is allowed to act as a *cNode*.

If a given *vNode* is in fact a malicious node, it could lie about integrity of the *cNode* it watches. To prevent that, the cluster head must receive multiple reports (their number exceeding a predetermined threshold) from distinct *vNodes* before actually considering a *cNode* as compromised. To some extent, this also makes the scheme resilient to errors from the *vNodes*.

In that way, nodes are allowed to act as *cNodes* only if they announce plausible amounts of residual energy. Assuming that this role consumes more energy than sensing only, the nodes elected as *cNodes* will sooner or later see their residual energy drop below the reserve of normal sensing nodes, which implies that they will not get re-elected at the next election. Note that the cases 2 and 3 make a compromised node decrement its announced energy as the time goes by. Even if inconsistency may be noticed and the compromise detected, this simple behavior ensures that the rogue node will stop being elected at some point in time.

Thus, the interest of *vNodes* can be summarized as follows: a compromised node cannot ensure the takeover of the *cNode* role at each cycle without cheating when announcing residual energy, and hence being detected by the *vNodes*. Detecting rogue *cNodes*, or forcing them to give up their role for later cycles, are the two purposes of the *vNodes*. The *vNode* role does not prevent a node from processing to its normal sensing activity (requests to *cNodes* must not occur too often, or too much power will be drained from the *vNodes*). The state machine of the nodes is presented in Figure 3.

Mathematical model for energy consumption A possible mathematical model that the cluster heads may use for computing the energy consumed by the nodes based on received observations is Rakhmatov and Vrudhula's diffusion model [46]. It provides a pretty accurate approximation of real consumption, taking into account chemical processes internal to the battery such as rate capacity effect and recovery effect. Rakhmatov and Vrudhula's diffusion model refers to the chemical reaction happening inside the battery electrolyte, and is summarized by equation (1):

$$\sigma(t) = \underbrace{\int_0^t i(\tau)\,\mathrm{d}\tau}_{l(t)} + \overbrace{\int_0^t i(\tau)\left(2\sum_{m=1}^{\infty}\exp^{-\beta^2 m^2 (t-r)}\right)\mathrm{d}\tau}^{u(t)} \tag{1}$$

where:

- $\sigma(t)$ is the apparent charge lost from the battery at t.
- $l(t)$ is the charge lost to the load ("useful" charge).
- $u(t)$ is the unavailable charge ("lost in battery" charge).
- $i(t)$ is the current at t.
- $\beta = \dfrac{\pi\sqrt{D}}{w}$, where D is the diffusion constant and w the full width of the electrolyte of the battery.

In practice, computing the first ten terms of the sum provides a good approximation.

5.3. Cluster coverage in case of heterogeneous activity

Deterministic election of the *cNodes* does not only introduce a flaw that compromised nodes could try to exploit. There is a second problem, independent from the nodes' behavior, which could prevent the detection of compromised nodes. If a region of the network happens to produce more traffic activity than the other parts of the network, the energy of its nodes will be drawn faster. In consequence, none of the n nodes with the highest residual energy (n being the desired number of *cNodes* during each cycle) will be located inside this region, and some nodes may not be covered for surveillance as long as traffic does not fade, possibly for all cycles. Figure 4 illustrates this problem.

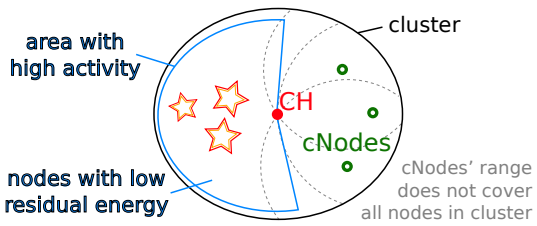

Figure 4. Illustrative scheme: *cNodes* are elected inside the area with less activity (thus with more residual energy) and do not cover nodes from the opposite side of the network

To address this issue we need to ensure that every node in the network is covered by at least one *cNode*. So the election process we presented in subsection 5.2 needs to be modified. The correct version is as follows:

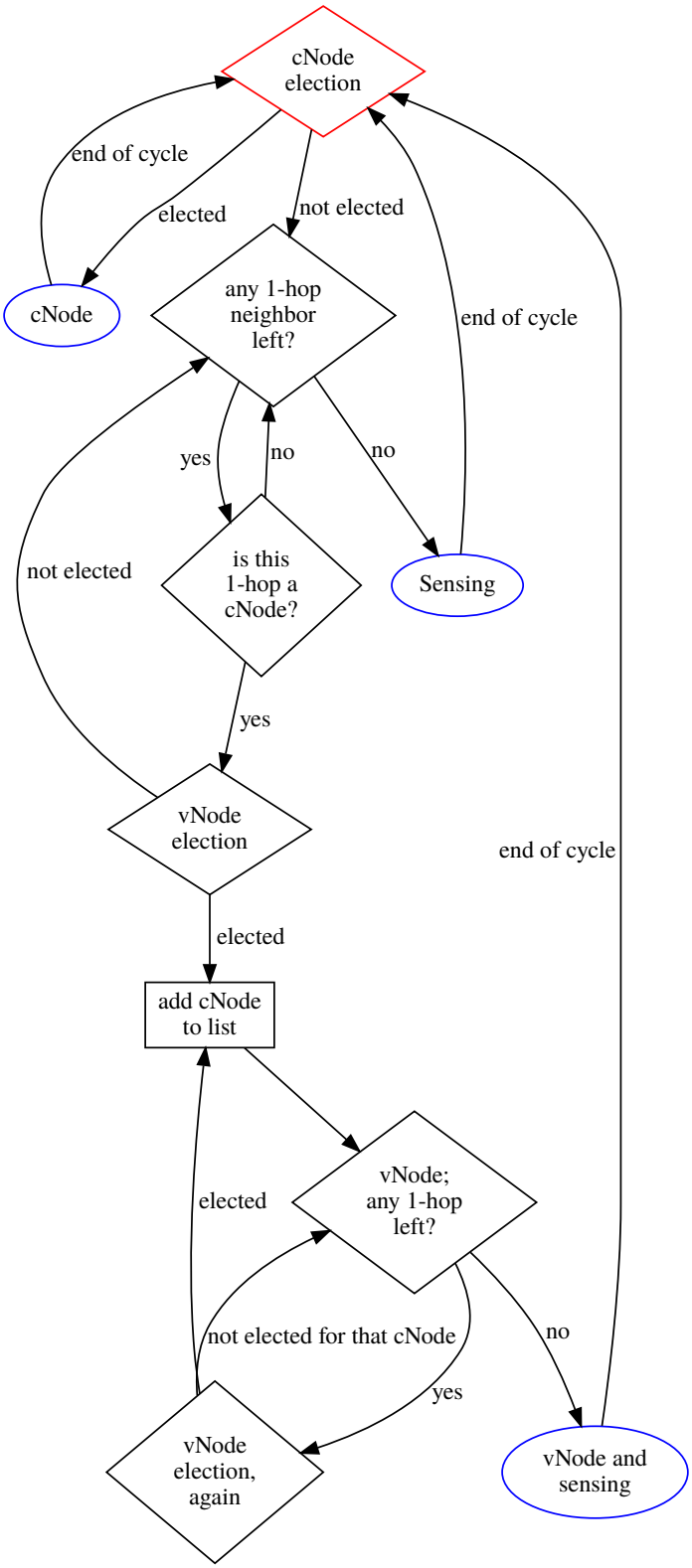

Figure 3. State machine of the (non-CH) nodes

1. During the first step, each node evaluates its residual energy and broadcasts the value;
2. The cluster head listens to all values. Other nodes also register all messages they hear into memory;
3. All nodes send to the CH the list of their 1-hop neighbors[‡];
4. The CH picks the n nodes among those with the highest residual energy, such that the n nodes cover all other nodes in range. If needed, it selects some additional nodes to cover all the cluster;
5. The CH returns a message to selected nodes to assign them the role of *cNode*.

Note that some clustering algorithms (such as HEED [9], for example) provide other election mechanisms (for cluster heads, but that can also be used for selecting *cNodes*) based on residual energy. We do not want to use it because energy only takes part in the process as a factor for the probability that the nodes declare themselves elected. Instead we prefer nodes to broadcast their residual energy in order to enable surveillance by the *vNodes*.

This second *cNodes* selection method is based on residual energy and intuitively brings a better balance to the network; but adding a second surveillance role (*vNodes*) adds to the complexity of the process, and increases the global consumption as well. A better approach might reside in reusing the observations made by the *cNodes* to elect the next set of monitoring nodes.

6. DEMOCRATIC ELECTION

Our third selection mechanism is called a democratic election, because it makes sensors vote for the nodes that will be selected as *cNodes*.

To detail this process, we will adopt the following notation. Let the symbol $\mathcal{N}$ denote the set of all nodes in the network and $\mathcal{CN}$ denote the set of *cNodes*. i is the index ranging over $\mathcal{N}$. Let RE_k be an array of residual energies of nodes reported by *cNodes* to the cluster head at the k^{th} iteration. The symbol $Obs_k[j]$ denotes an array containing observations made by *cNode* j on communications of its neighbors.

The election process starts with an initialization phase and then enters a loop block which is iteratively performed as long as the network is running.

6.1. Initialization phase

At the start of the process, the following actions are undertaken:

- Each node i in the network sends to the cluster head the value of its residual energy which is stored into a related array $RE_0[i]$.
- Each node acts as a *cNode* and starts controlling its neighbors. It keeps collecting and forwarding data (otherwise there would be no traffic to observe). Since the set of *cNodes* contains all nodes of the network, each node starts recording the communications made by its neighbors.
- The cluster head sets the counter of iterations k to 1.

6.2. Loop block

At each iteration k, the election process executes the next steps.

1. The duration of this surveillance step at any iteration k is random to prevent compromised nodes to simulate the behavior of sane nodes as long as possible. During step 1, each *cNode* controls the neighboring nodes (including other *cNodes*) by recording and adding up sizes of all packets sent or received by these nodes. Recall that all nodes in the first iteration ($k = 1$) are *cNodes*.

[‡]We do not deal with the case of compromised nodes cheating at this step of the process. Indeed they could announce extra virtual neighbors to try to escape from coverage.

2. At the end of iteration k, the cluster head (CH) asks each node i to send its residual energy value $RE_k[i]$ and asks each *cNode* j to send the array $Obs_k[j]$ containing its observations over transmission rates of its neighbors.
3. For each node i, the cluster head performs an analysis work as follows:
   - According to an adequate mathematical model, the CH assesses the energy consumption ECa related to the maximum of rates $\{Obs_k[j][i]\}|_{j \in \mathcal{CN}}$ observed by neighboring *cNodes* j during the current iteration k.
   - The CH also computes the value of the energy consumption ECd as the difference between residual energies declared in the two last steps (*i.e.*, $RE_k[i]$ and $RE_{k-1}[i]$).
   - If $|ECd - ECa| \leq \epsilon$ (where ϵ represents a tolerated error) then the node i is declared as sane and put into the set SEN of nodes eligible to take on a *cNode* functioning mode. Otherwise, it is removed from the set SEN of sane nodes (if it was there) and put into the pool SSN of suspicious nodes.
   - Let $SSN[i]$ stand for the number of times it has declared as suspicious from the start of the process. If $SSN[i] \geq threshold$ then the node is declared as compromised and put into a quarantine list. On the other hand, if a suspicious node has continually been declared as sane (more than some number of times) then it could be removed from SSN and put again into SEN.
4. Once step 3 is finished, the CH selects *cNodes* from the set SEN of eligible nodes in such a way that every node is controlled by at least two *cNodes*. Hence, as a *cNode* will be controlled by other *cNodes*, its misbehavior would be reported to the CH. For further iterations, such a rule helps detect and discard compromised nodes which have been chosen as *cNode* because they normally behaved in the past iterations and made false statement about their residual energies, unless they continue to undertake normal communications.
5. The process increments the number k of iterations and continues its iterative execution by going back to step 1.

6.3. Selecting cNodes among set SEN

At step 3 of the fair election process, cluster heads select the *cNodes* for the running iteration among the nodes inside SEN sets. Note that any selection criterion could be used at this point. For instance, *cNodes* could be randomly picked among elements of SEN until we have:

- enough *cNodes* (according to user's choice)
- and all nodes covered by at least two *cNodes*, as mentioned above.

Some other selection criteria could include:

- residual energy of the nodes
- connectivity index (number of direct neighbors)
- signal power
- *et cætera*

Several criteria can even be combined to obtain a weighted score, such as for instance in equation 2:

$$s_k[i] = (\alpha \times RE_k[i]) + (\gamma \times ci_k[i]) + (\delta \times sp_k[i]) + (\zeta \times nsl_k[i]) \tag{2}$$

where:

- $s_k[i]$ denotes the score for node i at iteration k.
- $RE_k[i]$ remains the residual energy for said node and iteration.
- $ci_k[i]$ would be the connectivity index of i.
- $sp_k[i]$ is the average signal power as perceived by the neighbors of node i.

- $nsl_k[i]$ is the maximum value between a predetermined integer value and the number of iterations before k since node i was last selected as a *cNode*.
- α, γ, δ and ζ would be constants fixed by the user.

This formula could be used to sort nodes in set SEN so that the cluster head can select the best possible *cNodes* in regard to retained criteria.

Setting the criteria and the weights for the formula is up to the user of the network. It should be adapted to the exact application and environment of the WSN. For instance it may be worth noting that for networks made of static nodes, the connectivity index and the signal power of the nodes are not expected to change much between two consecutive iterations. Therefore their weights should not be too high (in regard to the other weights in the formula) so as to avoid selecting the same *cNodes* at each iteration. In clustered networks, where all the nodes can reach their CH in a one-hop fashion, index connectivity or signal power might not even be relevant (once again, depending also on the deployed application). But even if we only work in clustered networks in this study—because they allow energy savings and a much better scaling—we should nevertheless consider other architectures in that respect, for not all WSNs are clustered. And many ones also work with mobile nodes. In such cases, evaluating the density of nodes or the quality of the links in the area where the candidate *cNodes* are located becomes more interesting because it generally has a higher impact on performances. Such observations remain valid for the chosen constraint stating that each node must be watched over by at least two *cNodes*. While it is a good thing in clusters where nodes are all gathered around their cluster head, it could be much harder to obtain in some other topologies. In a star-like network for instance, where most sensors would be on branches and only have two neighbors (one closer, one farther from the base station), it could result in all nodes being selected as *cNodes*.

Remark on Limit cases A WSN usually consists of a huge number of sensors scattered over a geographic zone. These sensors are grouped into clusters in such a way that the latter contain more than two or three sensors in order to fulfill their monitoring functions. Note that even if a cluster has only two nodes then the method will proceed as expected. After electing one of the cluster sensors as a cluster head (CH) a remaining node would be a cNode until the next round of CH election. Another solution might be to set a quorum of the cluster nodes which should be reached, otherwise cNode selection shall be deactivated. As the method proposes to select at least two cNodes then the failure of a cNode will not severely alter the control task. In the next step, the failing node will be discarded from the selection process as long as it fails to send its residual energy to the CH. However, if the cluster consists of two or three nodes, the failure of the cNode makes it needless to replace it with another one because the resulting cluster would be ineffective.

7. NUMERICAL RESULTS AND COMPARISON OF THE THREE METHODS

7.1. Setting up simulations

Now that we have defined three distinct selection processes for *cNodes*, we present numerical results obtained by simulation so as to compare the algorithms and to evaluate their relative performances. The `ns-2` software was used for this task. The simulations are based on a square grid of one hundred sensors, plus the cluster head (located at the centre of the square). We assume that all sensor nodes are using the same communication range. See Figure 5

Table I sums up the main parameters that were used for the simulations.

Evaluation of the detection mechanism itself (*e.g.,* in function of the number of *cNodes* in the cluster) has been realized in previous works, and it is not discussed again in this article [14].

Here the main purpose of the simulations is to compare the three selection methods. Hence a running a distinct "scenario" consists in choosing:

- one of those three mechanisms;

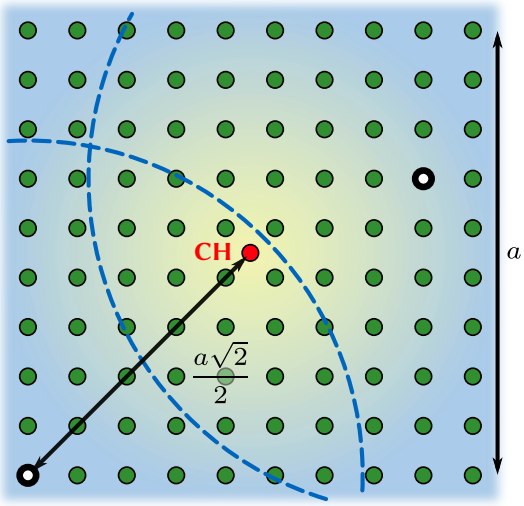

Figure 5. A 10×10 regular-grid cluster of size a

Table I. Simulation parameters

PARAMETER	VALUE
Simulation length	3,600 seconds
Number of sensors	100 (+ cluster head)
Number of *cNodes*	7–10 (depending on scenario)
Number of compromised nodes	1
Frequency for renewing the *cNodes* set	every 10 seconds
Length of initial phase for democratic election	60 seconds
Mobility of the nodes	null

PARAMETER	VALUE (NORMAL SENSORS)	VALUE (COMPROMISED SENSOR)
Emission rate	1 kB/s	35 kB/s
Packets length	500 bytes	100 bytes
Interval	random (Poisson)	constant
Consumption for emission	0.660 W	0.660 W
Consumption for reception	0.395 W	ignored
Initial energy amount	10 J — ∞ (dep. on sc.)	∞

- the number of *cNodes* that will run for every period;
- the initial energy amount for the sensors.

All graph display, for a given energy amount, the values obtained with five distinct scenarios. To refer easily to these scenarios, we will name them as follows:

|Stat10|
Ten *cNodes*, no renewal at all ("static" *cNodes* are selected in a random fashion just once, at the beginning of the simulation)

|Rand10|
Ten *cNodes*, with periodical renewal based on a random selection (first method); random selection itself is performed by cluster head

|ResE10|
Ten *cNodes*, periodical renewal based on residual energy (second method)

|Demc10|
Ten *cNodes*, periodical renewal based on democratic election (third method)

|Demc07|
Seven *cNodes* only, periodical renewal based on democratic election again (third method); changing the number of *cNodes* is done in order to observe its influence on the global consumption

All numerical results displayed on graphs are average values computed from the results obtained from ten distinct instances (for each scenario). These instances differ by the seed provided to the pseudorandom number generator of `ns-2`, and to the one used by the nodes for random selection processes.

The average energy consumption is computed from the values of the ninety-nine "sane" nodes, so they do not include the consumption of the cluster head and of the compromised sensor. Those two special nodes were provided with an unlimited energy amount for simulations:

- For cluster head: because it is mandatory to the functioning of the cluster, and because on a real instance, emptying its battery would trigger an immediate renewal of the CH. Furthermore, it receives all useful data sent by the nodes, so its energy consumption is high, and it would bend the average values we need to study here.
- For the compromised node: because we do not want it to stop emitting during the scenario; and again because the many packets it emits makes it consume a lot, which could bend the average values we need to observe. Also it is worth noting that this could be a real case scenario, if an attacker adds a more powerful node to the network (*e.g.,* a laptop). In such case, the node might not abide by the same energy rules.

7.2. Numerical results

7.2.1. Energetic consumption A first series of simulation instances were executed with unlimited available energy for the sensors (with regard to the length of the simulation, meaning the nodes could not empty their battery during these instances). It makes it possible to observe the average energy amount that was consumed by sensors with each selection method for the *cNodes*. The results are available on Figure 6. They indicate a similar consumption for methods |Stat10| and |Rand10|, because the random renewal of the *cNodes* does not cause by itself a high increase. Methods |ResE10| and |Demc10| consume more energy because of the need to gather energy from the nodes, and because of additional mechanisms used to secure the selection process.

Method |Demc10| costs more at the beginning of the simulation, as we can see by zooming in on the graph close to its origin, as in Figure 7. For the two scenarios based on democratic election, the consumption rises very quickly during the first minute: this is due to the initial phase (when all sensors watch over their neighbors in addition to performing actual sensing). But there is a reversal between the consumption of |ResE10| and |Demc10| shortly before reaching 40 minutes, given that with a longer delay, |Demc10| consumes less energy than a method using *vNodes*. As for |Demc07|, it is the least consuming method. This highlights the impact of the number of *cNodes* on the global energy consumption in the cluster: the fewer the *cNodes*, the more energy we save.

So methods taking residual energy into account (|ResE10|, |Demc10|) are, for a given number of *cNodes*, more demanding energywise. But it is also worth noting the load balancing they bring to the cluster. Figure 8 shows the standard deviation regarding energy consumption for the same simulations. We can observe that |Stat10| creates an increasing difference (in a linear fashion). Since *cNodes* are always the same sensors, they are always the same assuming the more demanding task, thus increasing

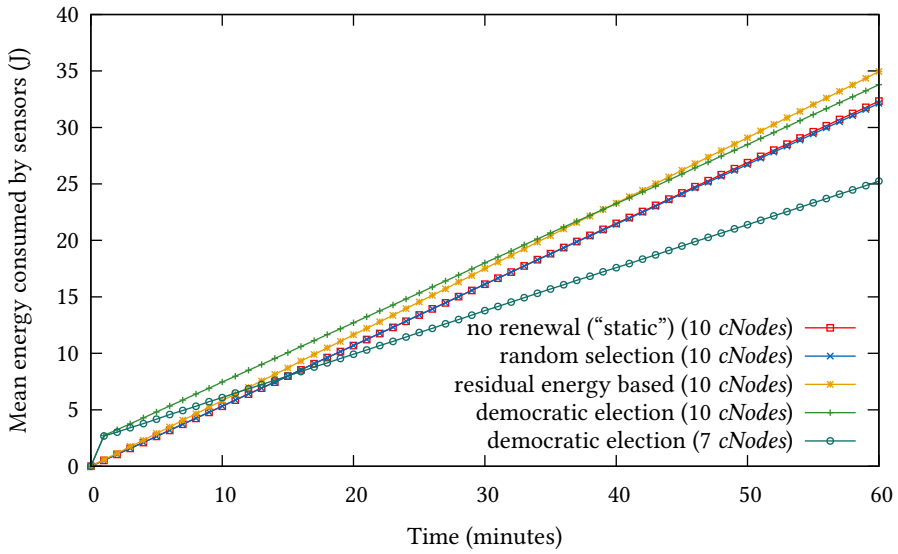

Figure 6. Average energy consumption for the sensors in function of elapsed time (initial energy amount: ∞)

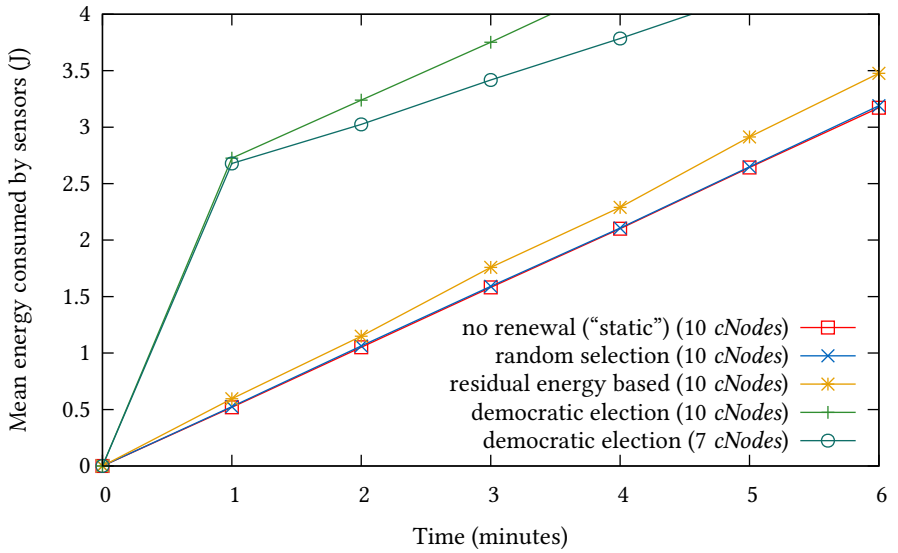

Figure 7. Average energy consumption for the sensors in function of elapsed time: zoom on the origin (initial energy amount: ∞)

the deviation with the other nodes. With |Rand10| method, the standard deviation rises quickly, but slows down and reach a stable value (approximatively 6 J at the end of the simulation, not displayed on the graph). This is because all sensors are selected approximatively the same number of time over a long simulation delay, due to the law of large numbers. And yet the standard deviation remains very high in comparison with those of |ResE10|, |Demc10| ans |Demc07|, all of them ensuring an excellent balancing of the energy consumption inside the cluster—which was their initial objective.

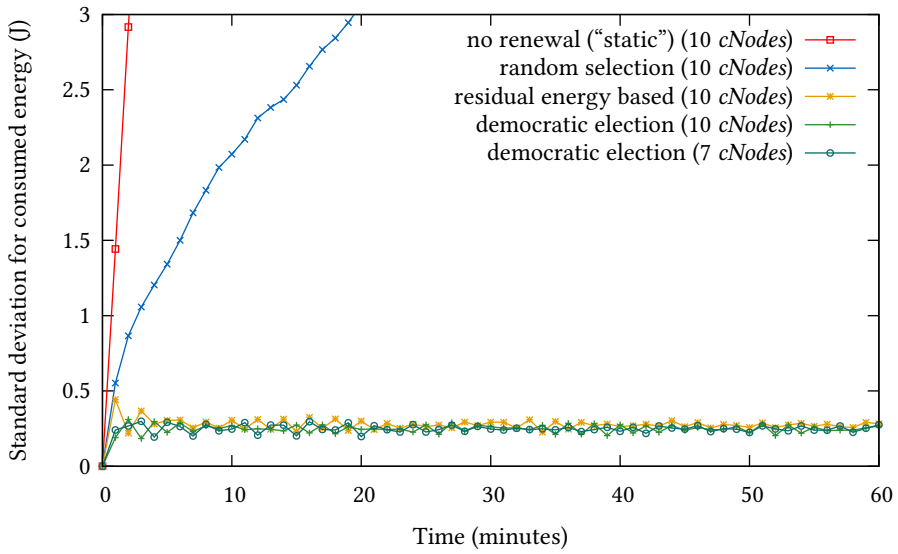

Figure 8. Standard deviation for the energy consumed by sensors in function of elapsed time (initial energy amount: ∞)

7.2.2. Network lifetime Two other sets of instances has been used to analyze the network lifetime. The evolution of the number of remaining nodes has been observed for instances with 10 J (first set), and then with 20 J (second set) for the initial energy amount of the sensors.

Figure 9 shows the number of nodes in activity depending on time, for a 10 J initial energy stock. |Stat10| method is the one that preserves the nodes for the longest duration, and by far. This is because

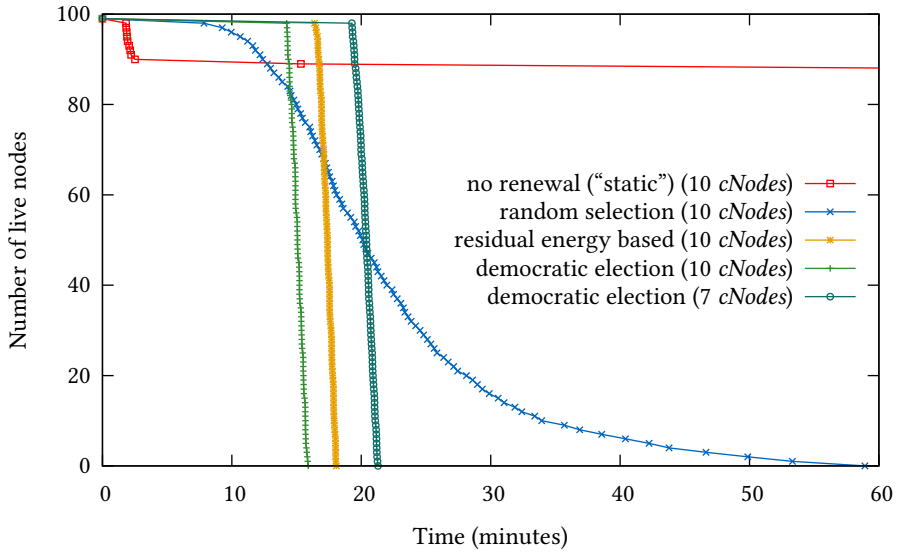

Figure 9. Number of nodes still in activity in function of time—Initial energy: 10 J

the *cNodes* are not renewed, and once they are "dead" there are no more high-consuming task running in the cluster [§]

The random |Rand10| method is efficient to preserve energy inside the cluster. As the number of selected *cNodes* is statistically a percentage of the global number of nodes inside the cluster (we select k nodes in a random fashion, without checking whether they are still alive or not), reducing the number of nodes does not produce any increase in the global consumption. In addition, the method is simple to set up and produces little overhead.

|ResE10|, |Demc10| and |Demc07| are not as efficient for energy preservation. The consumption is higher, be it because of the initial phase (|Demc10| and |Demc07|) or because of the use of *vNodes*. Thus the first node dies sooner than with |Rand10|, and the other nodes follow very quickly:

- because the standard deviation for residual energy is very low inside the cluster, so when the battery of a node reaches zero, it means that the over batteries are at a very low level as well;
- and because selecting as *cNodes* the 10 nodes with the most residual energy ensures that 10 *cNodes* are always in use, preventing the accidental selection of dead nodes. As a consequence, when first nodes die, the percentage of *cNodes* inside the cluster grows, increasing the average consumption for remaining nodes and emptying their batteries quickly.

Of the last three methods, we can observe that |Demc07| is the most efficient here (because it uses fewer *cNodes*), whereas |Demc10| is the least efficient because of the gap created by its initial phase. But this gap can be filled: this is what is shown by Figure 10, which comes from the execution of similar scenarios, but with 20 J (instead of 10) as initial energy amount. |Demc10| and |ResE10| consume

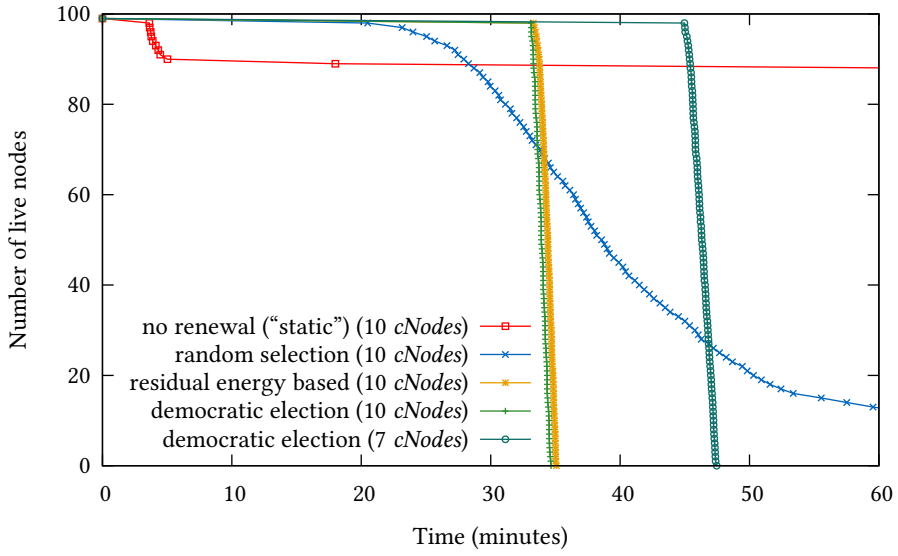

Figure 10. Number of nodes still in activity in function of time—Initial energy: 20 J

a nearly equivalent amount of energy in this case. The |Stat10| and |Rand10| methods in regard to Figure 9, but the nodes last twice as long. |Demc07| catches up (in efficiency) with |Rand10|, as this higher initial amounts enable the nodes to run for a longer duration and to soften more the increase due to the initial phase. With 10 J of initial energy, the nodes die soon after 20 minutes with |Demc07|, while

[§] The tenth *cNode* appears to empty its battery long after the ninth in those simulations. This is because we have average values compiled from several instances. On some executions, it may happen that the compromised node is randomly selected as one of the *cNode*, but we ignore its consumption and lifetime on the graph, so for those few instances it is as if we had nine *cNodes* only, and the "tenth" node to empty its battery does so much later.

|Rand10| still has half of its sensors alive. But with 20 J of initial energy, nearly three quarters of the nodes are dead for the |Rand10| method when the number drops for |Demc07|, little before 50 minutes. With an initial energy amount set to an even higher value, the difference of the number of *cNodes* would make the |Demc07| more efficient for the global consumption.

7.2.3. Detection rate The number of *cNodes* able to detect the attack is directly related to the number of nodes still in activity inside the cluster, since nodes running out of power can no longer perform the monitoring task. Figure 11 shows the average detection rates in time in simulation instances where a 10 J-high initial energy amount has been awarded to the nodes. The detection rate soon drops to zero

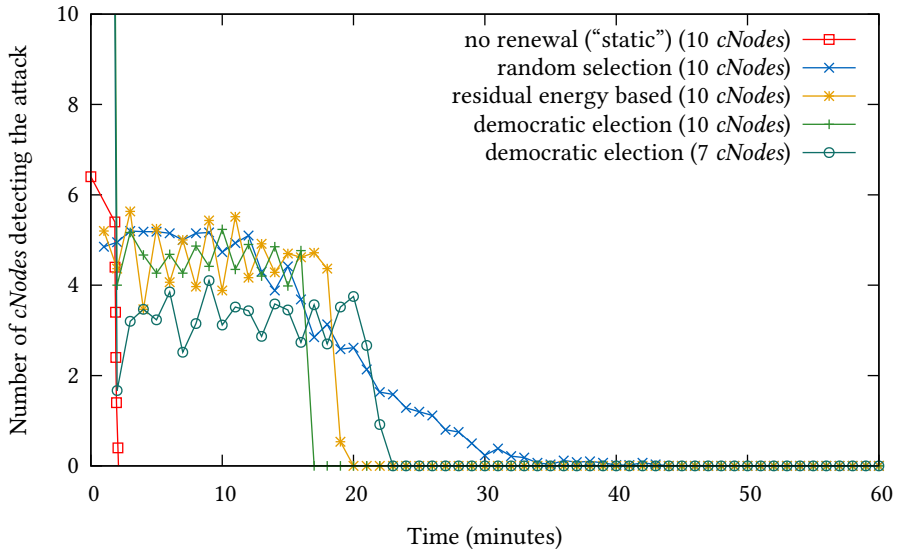

Figure 11. Detection rate of the attack in function of time—Initial energy: 10 J

with the |Stat10| method: once the initially selected *cNodes* run out of battery power, there is no other node able to detect the attack. |Rand10|, |ResE10| and |Demc10| all have a good detection level, with 5 *cNodes* on average detecting the attack—as long as most nodes remain in activity. |Demc07| method has fewer nodes detecting the attack, usually between 3 or 4 *cNodes*, but for a total of 7 *cNodes* in activity only. Detection runs over a longer duration than for |ResE10| and |Demc10|, given that nodes consume less on the whole. But it does not competes well with the |Rand10| method, which has nodes detecting the attack after 30 minutes of simulation.

It is worth noting that the position of the compromised node has a direct influencer over the number of nodes detecting the attack. Its chance to get detected depends on the number of its neighbor sensors indeed. For all simulation results provided in this section, the compromised node lays at the location depicted on Figure 12, and it has got 61 direct neighbors (excluding the CH), which represent as many potential *cNodes*[¶]. However monitoring a sensor does not systematically equals to detecting the bad behavior of this node (for example, if many collisions occur and the *cNode* fails to register all packets sent by compromised sensor).

This remark makes it possible to evaluate the probability that a rogue node has to escape monitoring (*i.e.*, to be watched by no *cNode*). With |ResE10|, |Demc10| and |Demc07| this probability is null as long as there are enough nodes remaining in the cluster to enforce the constraint stating that all sensor

[¶] The physical layer model used for the simulations does not have any random component: besides collisions, two nodes are either within mutual transmission range without any loss due to weak signal power, or too far away to communicate at all through direct transmission.

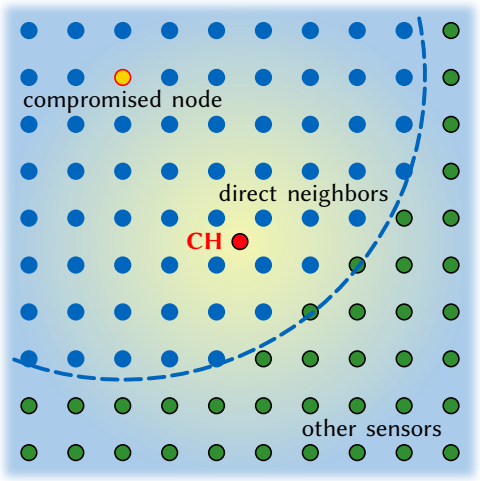

Figure 12. Scheme of the cluster showing the location of the sixty-one direct neighbors (in blue) of the compromised sensor (yellow node) as defined in simulations

must be covered by at least two *cNodes*. But the |Rand10| process is such that on some periods, all *cNodes* could be elected among the complementary set to the direct neighbors of the compromised node (that is, the set of green nodes on Figure 12). Let us compute this probability. With |Rand10|, *cNodes* repartition follows a hypergeometric distribution. We can use the following formula to compute the probability of k *cNodes* among the $n = 10$ to choose are selected among the $n = 61$ direct neighbors of the compromised sensor, for a total of $N = 100$ candidates:

$$P(k) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}} = \frac{m!}{k!\,(m-k)!} \times \frac{(N-m)!}{(n-k)!\,((N-m)-(n-k))!} \times \frac{n!\,(N-n)!}{N!}$$

Hence the probability of the compromised sensor not being monitored by any *cNode* is of:

$$P(k=0) \approx 3,6 \times 10^{-5} \approx 1/27228$$

It is extremely low, but again, we have to keep in mind that "monitoring" is not equal to "detection"

Figure 13 shows the same result as the previous graph, but for an initial energy of 20 J. As for the cluster lifetime, one can observe that the increased energy enabled the |Demc10| and |Demc07| to compensate for their initial phase and to gain improved performances over |ResE10|. |Demc10| slowly catches up with the |Rand10| method, but it will not reach equal performances since the democratic election has a slightly higher data control overhead.

After studying the performances and characteristics of the different selection methods, we can now highlight their pros and cons and provide recommendation depending on the configuration of deployed network.

8. CHOOSING A SELECTION PROCESS

8.1. Choosing a process depending on the application

Several methods have been proposed in this article to select the *cNodes*, and their performances were evaluated through software simulations. Here we sum up their properties and state what method is the most adapted to a given case. Actually there is no such thing as a "perfect method", each one has its pros and cons.

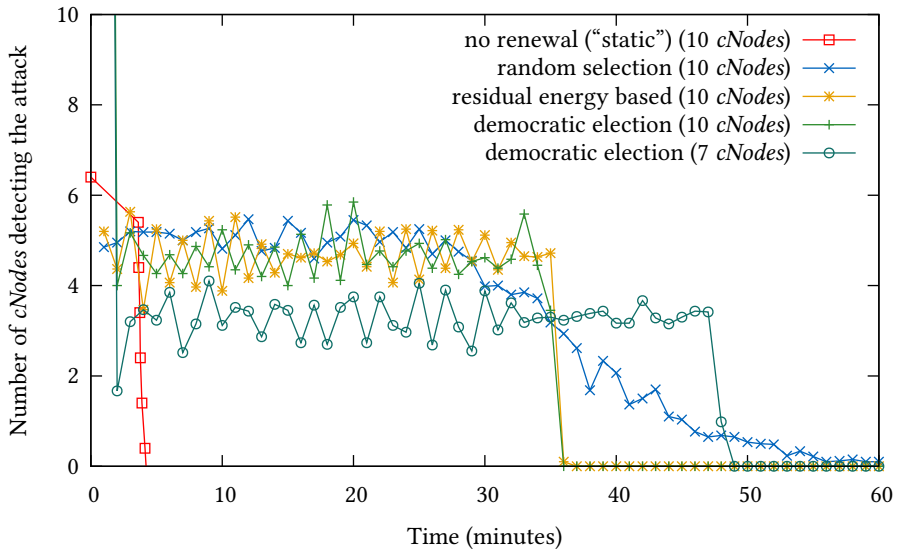

Figure 13. Detection rate of the attack in function of time—Initial energy: 20 J

No renewal of the *cNodes* Not renewing the *cNodes* makes it possible to preserve the battery of the sensors for a longer duration, but it does not bring a good security level to the network. The initial *cNodes* are soon to run out of power and to fall off. In addition, if the selection is performed in such a way that some nodes in the cluster are not covered by at least one *cNode*[‖], then these nodes will never get monitored. Thus setting up *cNodes* with no renewal is discouraged, unless *cNodes* have been determined by the operator prior to network deployment and they dispose of better hardware (including a better battery). In all other cases, one should rather not select any *cNode* at all instead of deploying a weak and short-lasting security mechanism.

Periodical renewal: random selection The random based selection process is a good compromise between security and network lifetime. This method is easy to set up. It brings little control data overhead, and consumes little energy (beside the necessary increase caused by the monitoring task). When the first nodes run out of power, the average number of *cNodes* is adapted so that the desired percentage of *cNodes* remains stable. The load balancing is not as good as with other methods though, and the application running in the network must be able to cope with the fact that some nodes will exhaust their batteries sooner than the others. The security level of this process is not absolute, since some sensors can be located outside the range area of *cNodes* for a given period. However the continual renewal of the *cNodes* and the laws of probability should ensure that a compromised node should be detected after a few periods at most.

Periodical renewal: selection based on residual energy The use of *vNodes* improves the security of the selection process based on residual energy, but it makes the nodes consume a high amount of energy. The balancing of the consumption inside the cluster is excellent, but it does not overtake the performances obtained with the democratic election process, which consumes less on the whole (once it has caught up for its initial phase). The only use case of this method would be a network where residual energy equilibrium between sensors is important, but where sensors would only have short

[‖] Or actually by the minimum number of *cNodes* required to have a sensor convicted, since cluster head should not take their decisions based on notifications from a single *cNode*.

periods of activity, for instance, where all sensors would run for a few minutes, perform measurements, and then reach an hibernation state for several hours. For all other cases, the democratic election is a better solution.

Periodical renewal: democratic election The performances of the democratic election method are similar to those of the selection based on residual energy in regard to the load balancing in the cluster, which means that they are excellent. But the overall consumption is better, at least once the initial phase has been compensated. However it remains far above the consumption reached by the random selection process. The constraint enforcing coverage of all nodes by at least two *cNodes* ensures a high monitoring ability, and the security of the mechanism is guaranteed by the "votes" of the *cNodes* helping their cluster head to designate their successors. This method has another advantage; by reusing the data collected through nodes monitoring, it is possible to use other factors to select the nodes. These may include signal power, connectivity index of trust value. This method is recommended for applications with strong constraints on load balancing, and for which the entirety of the set of sensors must remain alive as long as possible.

Pros and cons of each method are summed up in Table II. As for Table III, it displays the use cases for those processes.

Table II. Pros and cons of the different selection methods

METHOD	ADVANTAGES	DRAWBACKS
No renewal	• Preservation of non selected nodes	• Bad monitoring
Random selection	• Moderate energy consumption • Easy to implement • The number of *cNodes* remains a percentage of the active sensors • Good turnover of the nodes; random process, which prevents the attacks	• Moderate balancing of the energy consumption inside the cluster • Some sensors could escape monitoring during some periods
Selection based on residual energy	• Excellent load balancing • All sensors are monitored by at least two *cNodes*	• Adding *vNodes* is a constraint; it costs more energy and makes the scheme more complex to implement • Consumes a lot of energy
Democratic election	• Excellent load balancing • All sensors are monitored by at least two *cNodes* • Moderate energy consumption (after initial phase) • Can be adapted to take other parameters into account for the selection	• Initial phase consumes a high amount of energy

8.2. Adapting the process to the network

8.2.1. Number of cNodes The numerical results presented in former section, and especially the relative performances of the two scenarios based on democratic election with seven or ten *cNodes*, highlight the importance of the number of *cNodes* in the cluster on energy saving. As the *cNodes* have a higher consumption, the more they are, the faster the energy is consumed in the network. But at the same time the probability to detect the attack increases. Thus it is interesting to observe what the optimal number of *cNodes* is so as to get a good coverage of the cluster without loosing too much energy. In our case, using only seven *cNodes* seems reasonable, in particular with methods that ensure a minimal coverage of all the sensors. With a lower number of sentries, the stakes of some sensors not being covered get high.

Table III. Proposed use cases for the selection methods

METHOD	USE CASE
No renewal	Highly discouraged in general (provided security is not worth loosing energy because of this scheme, unless dedicated hardware is used)
Random selection	Applications for which security is not the highest priority, and for which the operator wants to keep sensors alive as long as possible (but not necessarily *all* sensors)
Selection based on residual energy	One would better use the democratic election process, unless the cluster has very short periods of activity.
Democratic election	Applications for which security is essential, and where the entire set of sensors must be kept alive together for the longest possible duration. However we recommend against this method for applications enforcing very short periods of activity for the cluster.

The number of *cNodes* to use also depends of the network topology. As a reminder, we have been working in a cluster where all nodes are direct neighbors of the cluster head. If the *cNodes* are to be deployed on another topology, for example, without any cluster, there may be different coverage problems.

8.2.2. Constraints on cluster coverage The example of a star-like network was mentioned in Subsection 6.3, and is depicted on Figure 14. With such a topology, it is necessary to adapt (or to cancel) the rule stating that each sensor should be monitored by at least two *cNodes*: otherwise, all the sensors will be selected. And yet this rule remains very useful in clustered networks such as the one we have

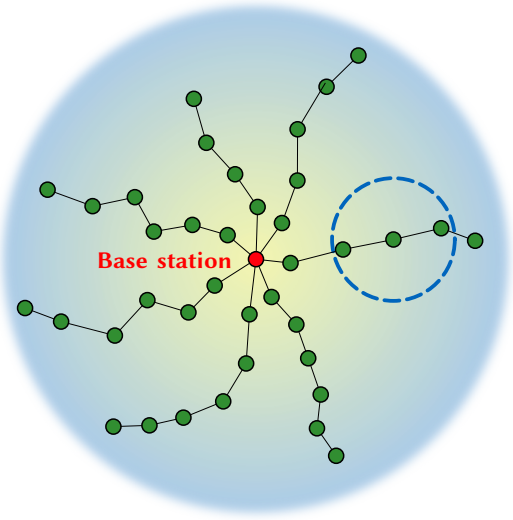

Figure 14. Scheme of a star-like network. The blue dashed circle represents the transmission range of a node.

been studying. Thus the operator needs to carefully study the expected topology of their network before decided whether to implement it.

8.2.3. Detection mechanism Setting up an efficient intrusion detection system relies on a rigorous implementation of a well-designed algorithm run by the *cNodes* to detect suspect behaviors. No particular recommendation is provided here about this algorithm, but it remains essential to establish

the comprehensive specifications of the network, including expected use cases and the value of variables for normal functioning, prior to deploying the network, so that to obtain the most accurate detection rules. Formal models of the system may also be relevant. We have proposed in other studies some formal modeling of the detection process [47] as well as of the interactions between legitimate and compromised sensors [48]

9. CONCLUSION

The use of monitoring nodes in a clustered wireless sensor networks represents an efficient and scalable method for detecting denial of service attacks. As monitoring entails a higher energy consumption for the sentry nodes, periodically renewing the set of *cNodes* is compulsory. So far there has been little work on the detailed implementation of the designation process for this set. In this paper we proposed three different methods to perform the selection of the monitoring nodes in an energy-efficient and secure way. Each method has its pros and cons: random selection is low consuming and easy to set up, but can lead to coverage issues; mechanisms based on residual energy of the nodes, including democratic election process, tend to provide a better load balance over the clusters, but the overall consumption gets higher because of additional security measures that are needed to prevent compromised sensors to monopolize the *cNode* roles. The selection processes have been extensively tested through simulations, and the numerical results obtained show that effective global consumption and energy balancing correspond to expectations. This leads to the description of the different recommended use cases for each selection process.

Future work leads include looking for other possible selection mechanisms so as to improve even more the consumption and the security of the scheme. Working with other detection methods, for example, by running other intrusion detection systems (not based on rules), would constitute an interesting approach as well. We also intend to vary the involved network applications so as to obtain clusters with different levels of activity (depending on geographical areas), as well as to use different network topologies (star-like clusters, or non-clustered networks for instance). All of these should, again, be tested with software simulations; but we also plan to use real hardware to implement our selection methods on a physical test bed.

Another angle of approach resides in the use of formal modeling tools, which can be used to represent the system and to extract properties and performance indices about it. In previous studies we have used models such as Markov chains, Petri networks, timed and hybrid automata, and game theory to analyze the selection process or the interactions between legitimate and compromised nodes, but these leads are to be further investigated. We have not been so far as running actual model checking tools, for instance.

References

1. Jalel Ben-Othman and Bashir Yahya. Energy efficient and qos based routing protocol for wireless sensor networks. *Journal of Parallel and Distributed Computing*, 70(8):849–857, August 2010.
2. Thibault Bernard and Hacène Fouchal. A low energy consumption MAC protocol for WSN. In *Proceedings of the 2012 IEEE International Conference on Communications (ICC'12)*, pages 533–537, Ottawa, ON, Canada, June 2012.
3. William R. Claycomb and Dongwan Shin. A novel node level security policy framework for wireless sensor networks. *Journal of Network and Computer Applications*, 34(1):418–428, January 2011.
4. Marcos A. Simplicio, Jr, Bruno T. de Oliveira, Paulo S. L. M. Barreto, Cintia B. Margi, Tereza C. M. B. Carvalho, and Mats Naslund. Comparison of authenticated-encryption schemes in wireless sensor networks. In *Proceedings of the 36th Annual IEEE Conference on Local Computer Networks*, pages 454–461, Bonn, Germany, October 2011.
5. Jalel Ben-Othman, Karim Bessaoud, Alain Bui, and Pilard. Self-stabilizing algorithm for efficient topology control in wireless sensor networks. *Journal of Computational Sciences*, 4(4):199–208, July 2013.
6. Fei Hu and Neeraj K. Sharma. Security considerations in ad hoc sensor networks. *Ad Hoc Networks*, 3(1):69–89, January 2005.
7. Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the IEEE 33rd Hawaii International Conference on System Sciences*, volume 8, Maui, HI, USA, January 2000.

8. Suat Ozdemir and Yang Xiao. Secure data aggregation in wireless sensor networks: A comprehensive overview. *Computer Networks*, 53(12):2022–2037, August 2009.
9. Ossama Younis and Sonia Fahmy. HEED: A Hybrid, Energy-Efficient Distributed clustering approach for ad-hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, October 2004.
10. Said Fouchal and Ivan Lavallée. Fast and flexible unsupervised clustering algorithm based on ultrametric properties. In *Proceedings of the 7th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet'11)*, Miami, FL, USA, November 2011.
11. Said Fouchal, Quentin Monnet, Djamel Mansouri, Lynda Mokdad, and Malika Ioualalen. A new clustering algorithm for wireless sensor networks. In *Proceedings of the 17th IEEE Symposium on Computers and Communications (ISCC'12)*, Nevşehir, Turkey, July 2012.
12. Yao Liang. Efficient temporal compression in wireless sensor networks. In *Proceedings of the 36th Annual IEEE Conference on Local Computer Networks*, pages 470–478, Bonn, Germany, October 2011.
13. Gu Hsin Lai and Chia-Mei Chen. Detecting denial of service attacks in sensor networks. *Journal of Computers*, 4(18), January 2008.
14. Malek Guechari, Lynda Mokdad, and Sovanna Tan. Dynamic solution for detecting denial of service attacks in wireless sensor networks. In *Proceedings of the 2012 IEEE International Conference on Communications (ICC'12)*, Ottawa, Canada, June 2012.
15. Ahmad Yusri Dak, Saadiah Yahya, and Murizah Kassim. A literature survey on security challenges in VANETs. *International Journal of Computer Theory and Engineering*, 4(6):1007–1010, December 2012.
16. Sahabul Alam and Debashis De. Analysis of security threats in wireless sensor network. *International Journal of Wireless and Mobile Networks*, 6(2):35–46, April 2014.
17. Soufiene Ben Othman, Abdullah Ali Bahattab, Abdelbasset Trad, and Habib Youssef. Confidentiality and integrity for data aggregation in WSN using homomorphic encryption. *Wireless Personal Communications*, September 2014.
18. Quentin Monnet, Lynda Mokdad, and Jalel Ben-Othman. Data protection in multipaths WSNs. In *Proceedings of the 18th IEEE Symposium on Computers and Communications (ISCC'13)*, Split, Croatia, July 2013.
19. Hai-Yan Shi, Wan-Liang Wang, Ngai-Ming Kwok, and Sheng-Yong Chen. Game theory for wireless sensor networks: A survey. *Sensors*, 12(7):9055–9097, July 2012.
20. Ping Guo, Jin Wang, Jiezhong Zhu, and Yaping Cheng. Authentication mechanism on wireless sensor networks: A survey. In *Proceedings of the 2nd International Conference on Information Technology and Computer Science (ITCS'13)*, volume 25, pages 425–431, Beijing, China, July 2013.
21. Ping Guo, Jin Wang, JieZhong Zhu, YaPing Cheng, and Jeong-Uk Kim. Construction of trusted wireless sensor networks with lightweight bilateral authentication. *International Journal of Security and Its Applications*, 7(5):225–236, September 2013.
22. Harjot Bawa, Parminder Singh, and Rakesh Kumar. An efficient novel key management scheme for enhancing user authentication in a WSN. *International Journal of Computer Network and Information Security*, 5(1):56–64, January 2013.
23. Leonidas Kazatzopoulos, Costas Delakouridis, and Christos Anagnostopoulos. WSN location privacy scheme enhancement through epidemical information dissemination. *International Journal of Communication Networks and Information Security*, 6(2):162–167, August 2014.
24. Chinnu Mary George and Manoj Kumar. Cluster based location privacy in wireless sensor networks against a universal adversary. In *Proceedings of the International Conference on Information Communication and Embedded Systems (ICICES'13)*, pages 288–293, Chennai, India, February 2013.
25. A. Varshovi and B. Sadeghiyan. Ontological classification of network denial of service attacks: Basis for a unified detection framework. *Scientia Iranica*, 17(2):133–148, December 2010.
26. Shio Kumar Singh, M. P. Singh, and D. K. Singh. A survey on network security and attack defense mechanism for wireless sensor networks. *International Journal of Computer Trends and Technology*, May 2011.
27. Mohammad Momani and Subhash Challa. Survey of trust models in different network domains. *International Journal of Ad Hoc, Sensor and Ubiquitous Computing*, 1(3):1–19, September 2010.
28. M. Carmen Fernández-Gago, Rodrigo Román, and Javier Lopez. A survey on the applicability of trust management systems for wireless sensor networks. In *Proceedings of the 3rd International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SECPerU'07)*, pages 25–30, Istanbul, Turkey, July 2007.
29. Mohammad Reza Rohbanian, Mohammad Rafi Kharazmi, Alireza Keshavarz-Haddad, and Manije Keshtgary. Watchdog-LEACH: A new method based on LEACH protocol to secure clustered wireless sensor networks. *Advances in Computer Science: An International Journal*, 2(3):105–117, July 2013.
30. Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5):521–534, September 2002.
31. Muhammad Hasan Islam, Kamran Nadeem, and Shoab A. Khan. Optimal sensor placement for detection against distributed denial of service attacks. In *Proceedings of the 2009 International Conference on Advanced Computer Control (ICACC'09)*, pages 675–679, Singapore, January 2009.
32. Jun-Won Ho. Distributed detection of node capture attacks in wireless networks. In Hoang Duc Chinh and Yen Kheng Tan, editors, *Smart Wireless Sensor Networks*, chapter 20, pages 345–360. InTech, December 2010.
33. Sudip Misra, P. Venkata Krishna, Kiran Isaac Abraham, Navin Sasikumar, and S. Fredun. An adaptive learning routing protocol for the prevention of distributed denial of service attacks in wireless mesh networks. *Computer and Mathematics with Applications*, 60(2):294–306, 2010.
34. Ju-Hyung Son, Haiyun Luo, and Seung-Woo Seo. Denial of service attack-resistant flooding authentication in wireless sensor networks. *Computer Communications*, 33(13):1531–1542, August 2010.
35. Amrita Ghosal and Sipra DasBit. A lightweight security scheme for query processing in clustered wireless sensor networks. *Computers and Electrical Engineering*, April 2014.

36. Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14–15):2826–2841, October 2007.
37. M. J. Handy, M. Haase, and D. Timmerman. Low energy adaptive clustering hierarchy with deterministic cluster-head selection. In *Proceedings of the 4th IEEE International Workshop on Mobile and Wireless Communications Networks*, pages 368–372, Stockholm, Sweden, 2002.
38. B. Brahma Reddy and K. Kishan Rao. A modified clustering for LEACH algorithm in WSN. *International Journal of Advanced Computer Science and Applications*, 4(5):79–83, May 2013.
39. Naveen Chawla and Ashish Jasuja. Algorithm for optimizing first node die (FND) time in LEACH protocol. *International Journal of Current Engineering and Technology*, 4(4):2748–2750, August 2014.
40. Leonardo B. Oliveira, Adrian Ferreira, Marco A. Vilaça, Hao Chi Wong, Marshall Bern, Ricardo Dahab, and Antonio A. F. Loureiro. SecLEACH — on the security of clustered sensor networks. *Signal Processing*, 87(12):2882–2895, December 2007.
41. Maryam Mohi, Ali Movaghar, and Pooya Moradian Zadeh. A bayesian game approach for preventing DoS attacks in wireless sensor networks. In *Proceedings of the 2009 International Conference on Communications and Mobile Computing (CMC'09)*, Kunming, Yunnan, China, February 2009.
42. Giuseppe Anastasi, Marco Conti, Mario di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, May 2009.
43. P. Sivakumar, K. Amirthavalli, and M. Senthil. Power conservation and security enhancement in wireless sensor networks: A priority based approach. *International Journal of Distributed Sensor Networks*, May 2014.
44. Quentin Monnet and Lynda Mokdad. Dos detection in wsns: Energy-efficient designs and modeling tools for choosing monitoring nodes. In Mohammad S. Obaidat, Faouzi Zaraia, and Petros Nicopolitidis, editors, *Modeling and Simulation of Computer Networks and Systems*, chapter 28, pages 795–840. Elsevier, April 2015.
45. Jinat Rehana. Security of wireless sensor network. Technical report, Helsinki University of Technology, 2009.
46. D. Rakhmatov and S. Vrudhula. An analytical high-level battery model for use in energy management of portable electronic systems. In *Proceedings of the International Conference on Computer Aided Design (ICCAD'01)*, pages 488–493, San Jose, CA, USA, November 2001.
47. Paolo Ballarini, Lynda Mokdad, and Quentin Monnet. Modeling tools for detecting DoS attacks in WSNs. *Security and Communication Networks*, 6(4):420–436, April 2013.
48. Quentin Monnet. *Mechanisms and Modeling Tools for Protection Against Denial of Service Attacks in Wireless Sensor Networks*. PhD thesis, Université Paris-Est, Créteil, France, July 2015. (In French).