Computer Communications 119 (2018) 66–82

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Whac-A-Mole: Smart node positioning in clone attack in wireless sensor networks

Wafa Ben Jaballah[*,a], Mauro Conti[b], Gilberto Filè[b], Mohamed Mosbah[c], Akka Zemmari[c]

[a] Thales, France
[b] Department of Mathematics, University of Padua, Italy
[c] LaBRI, Bordeaux INP, University of Bordeaux, CNRS, UMR 5800, France

ARTICLE INFO

Keywords:
Clone detection
Distributed detection
Smart node positioning

ABSTRACT

Wireless sensor networks are often deployed in unattended environments and, thus, an adversary can physically capture some of the sensors, build clones with the same identity as the captured sensors, and place these clones at strategic positions in the network for further malicious activities. Such attacks, called clone attacks, are a very serious threat against the usefulness of wireless networks. Researchers proposed different techniques to detect such attacks. The most promising detection techniques are the distributed ones that scale for large networks and distribute the task of detecting the presence of clones among all sensors, thus, making it hard for a smart attacker to position the clones in such a way as to disrupt the detection process. However, even when the distributed algorithms work normally, their ability to discover an attack may vary greatly with the position of the clones. We believe this aspect has been greatly underestimated in the literature. In this paper, we present a thorough and novel study of the relation between the position of clones and the probability that the clones are detected. To the best of our knowledge, this is the first such study. In particular, we consider four algorithms that are representatives of the distributed approach. We evaluate for them whether their capability of detecting clone attacks is influenced by the positions of the clones. Since wireless sensor networks may be deployed in different situations, our study considers several possible scenarios: a uniform scenario in which the sensors are deployed uniformly, and also not uniform scenarios, in which there are one or more large areas with no sensor (we call such areas “holes”) that force communications to flow around these areas. We show that the different scenarios greatly influence the performance of the algorithms. For instance, we show that, when holes are present, there are some clone positions that make the attacks much harder to be detected. We believe that our work is key to better understand the actual security risk of the clone attack in the presence of a smart adversary and also with respect to different deployment scenarios. Moreover, our work suggests, for the different scenarios, effective clone detection solutions even when a smart adversary is part of the game.

1. Introduction

Wireless sensor networks (WSN) are composed of a number of tiny, low-cost sensor nodes with limited resources. There are stationary and mobile networks: in mobile networks, sensors move, whereas in stationary ones, they remain still. In what follows, we only consider stationary networks. Such networks are often deployed in inaccessible areas in order to collect various information, such as environment monitoring and object tracking [1–9]. As sensor networks could be deployed in remote and hostile regions, they are unattended and could be attacked by adversaries [10–14]. In particular, since sensors normally are not equipped with tamper-resistant hardware, sensor networks are vulnerable to the **node clone attack** that works as follows. The adversary can capture one sensor node, read all information contained in it, in particular the identity (ID) and the cryptographic information, and fabricate many clones of the captured sensor. All clones have the same ID and cryptographic keys as the captured sensor. The clones are placed in strategic positions of the network for further malicious activities. Since all clones have valid credentials and can behave both as normal sensors and as devious attackers, it is difficult to detect them. In addition, clones can launch insider attacks potentially baleful for the functionality of the network [15,16]. This explains why researchers carried out a substantial amount of work for devising techniques that detect node clone attacks. Most of these techniques are

* Corresponding author.
E-mail addresses: wafa.benjaballah@thalesgroup.com (W. Ben Jaballah), conti@math.unipd.it (M. Conti), gilberto@math.unipd.it (G. Filè), mosbah@labri.fr (M. Mosbah), zemmari@labri.fr (A. Zemmari).

https://doi.org/10.1016/j.comcom.2018.01.010
Received 30 December 2016; Received in revised form 13 September 2017; Accepted 27 January 2018
Available online 06 February 2018

based on the **exclusiveness of sensor location** which is as follows: each honest sensor has a unique ID and a unique location in which it is, thus, when there are two distinct sensors that are in different locations and claim to have the same ID, they must be clones and a clone attack is being perpetrated. To discover when this situation occurs, all sensors must broadcast at regular intervals a location claim that contains their ID and location. These claims somehow arrive to some sensors, called witnesses, that have the task of checking whether they received two claims with equal ID and different locations. When a witness discovers such a case it *triggers the alarm*. We explain all these operations in successive sections. Here we focus on the main ideas.

The simplest way to exploit the above idea is to send all location claims to a unique central witness [1,4,6,17–20]. Clearly, if all claims arrive to this witness, then clone attacks are surely detected. However, such approach has some evident drawbacks. The central witness needs a lot of resources both for memorizing the information and for processing it. Moreover, sensors near to the central witness have to do a lot of communications and finally, since the attacker knows where the central witness is, he could place his clones in such a way as to prevent that sensitive information flows towards it. There are also centralized techniques based on other properties than the exclusiveness of sensor location, see [6] for more information. In our work, we do not consider the centralized approach and we focus on the distributed approach for detecting clone attacks. In the distributed approach, the role of witness is shared among all the sensors of the network. In this way the energy consumption is distributed through the network (thus, providing better scalability), and the adversary has an harder time to cover his attack by cleverly positioning his clones, because he does not know where the witnesses are (resiliency against smart attacks). In the literature different distributed algorithms have been proposed [15,19,21–27]. In [23] the authors present a comparative survey of centralised and distributed algorithms.

Considering the above mentioned advantages of distributed algorithms, we decided to focus on them. Our goal is to measure the impact that clone positions may have on the probability that the different algorithms detect the clone attack. We have chosen four algorithms that are meaningful representatives of the most interesting proposals: the Line-Selected Multicast (LSM), proposed in [19], the Randomized Efficient and Distributed protocol (RED), proposed in [15], the Single Deterministic Cell (SDC) of [21] and the Random Walk (RAWL) of [22]. In our opinion these algorithms incorporate the most relevant ideas for detecting clone attacks. Some of these algorithms, in particular RAWL, have been extended, for instance in [25] to improve energy and storage consumption. However, these extensions are not important for our study which focuses on detection probability.

More precisely, what we do is as follows. We study the behavior of the clone detection probability of LSM, RED, SDC and RAWL when these algorithms are used in different scenarios, i.e, with different configurations of wireless networks and with the clones placed in different positions. We have evaluated the performances of the four algorithms by means of the simulator VISIDIA [28–30]. We have considered the following three scenarios for the wireless network:

(i) a square on which the sensors are uniformly distributed;
(ii) a square with a central area that has the form of "H" on which there are no sensors and therefore communications are forced to *flow around the H*;
(iii) a square in which there are four round areas (each with a surface of about 1/20 of the square) with no sensors and thus communications must *flow around these areas*.

In what follows, areas of the deployment square without sensors are called **holes**. Thus in scenario (ii) we have a hole in the form of an "H" and in scenario (iii) we have four disjoint round holes. For each scenario we consider the presence of two clones. One clone is fixed in some special position (like the center of the square or one vertex of the square) and the other clone is in any other position on the square (but never in a hole). Thus, for each algorithm we evaluate whether there are smart ways of placing the two clones. The results we find in this way are a lower bound on the detection probability of all clone attacks. In fact an attack using more than two clones has higher probability to be detected than one with two clones only. Moreover, our analysis with two clones suggests positions in which clones are difficult to be detected. If the deployment of the WSN offers several such positions, the adversary may mount attacks with more than two clones that still have a good chance of not being detected.

Contributions. The main contributions of our work are as follows:

- We present in a common and formal framework the four algorithms, LSM, RED, SDC and RAWL that are representatives of the distributed approach to the detection of clone attacks. This uniform presentation shows very clearly differences and similarities of the algorithms.
- We compare the impact of clone positioning on the probability of detection of clone attacks by the four algorithms using the VISIDIA simulator [28–30].
- The results of the simulation reveals that the presence of holes in the network configuration has important effects. In fact, the simulation shows that such configurations offer hiding places for clones that are difficult to track even by the strongest algorithms. Moreover, we show that configurations with holes tend to improve the precision of the algorithms based on random paths (LSM and RAWL). On the contrary, they hinder the precision of the algorithms that decide the witnesses at the beginning of the protocol execution, such as RED and SCD. These facts are relevant for anyone designing new clone detection algorithms or managing wireless sensor networks.

Organization. The remainder of this paper is organized as follows. In Section 2 we explain all the characteristics of the wireless network and of the adversary that are assumed throughout the paper. In Section 3 we present the main ideas of the distributed approach to clone attacks detection. In particular, we describe the four algorithms LSM, RED, SDC and RAWL in a uniform and precise framework. In Section 4, we consider various scenarios, and we present the simulation results of these four algorithms. Section 5 concludes the work.

2. Network and adversary models

In the following, we present the network and adversary models in Sections 2.1 and 2.2, respectively. We consider distributed algorithms for clone attack detection, therefore, the models that we describe are the traditional ones for such algorithms.

2.1. Network model

The network is composed by sensors that are not tamper-resistant, have a limited power supply and also a limited transmission range. Each sensor has an *ID* and a pair of identity-based public and private keys, as described, for instance in [31]. Moreover, each sensor is aware of its own position by means, for instance, of a GPS system. We also assume that sensors are loosely synchronized. In particular, they are able to produce time stamps that the other sensors can check for freshness. With this data, at regular time intervals, every sensor computes and sends to its neighbors a **location claim**, that contains its *ID*, its location, a time stamp and a hash of all the previous data, signed with its secret key. If a sensor does not do this, it will be isolated by its (honest) neighbors as suspect of having been compromised by the attacker. When a neighbor receives a location claim, it checks that this

```
1  Procedure Broadcast();
2  /* Broadcast() is common to all algorithms                                   */
3  C = < this.ID, is_claim, this.LOC, time() > ;
4  SC = < C, this.K^priv(C) > ;
5  foreach L ∈ this.neighbors do
6      this → L: < this.ID, L, SC > ;
7  end procedure

8  Procedure Receive_LSM(M);
9  if is_claim(M) then
10     /* claim case                                                            */
12     < ID, −, SC > ← M;
13     < C, S > ← SC ;
14     if bad_signature (C, S) || incoherent_location(C) then
15         discard M;
16     do with probability p;
17         LOC = random(g);
18         /* LOC is a set of g locations                                       */
19         for all L ∈ LOC do;
20             FM = < ID, L, is_forward, SC > ;
21             this → GPRS (this.LOC, L): FM ;
22             end forall ;
24 else
25     /* forwarding case                                                       */
26     < ID, L, −, SC > ← M ;
27     < C, S > ← SC ;
28     if detect_clone(C) then
29         trigger revocation procedure for ID
30     else
31         put_in_store(C);
32         n = GPSR(this.LOC, L);
33         if n != this.LOC then
34             this → n : M;
35 end procedure
```

Algorithm 1. LSM.

```
1  Procedure Receive_RED(M) ;
2  if is_claim(M) then
3      < ID, −, SC > ← M;
4      < C, S > ← SC ;
5      if bad_signature (C, S ) || incoherent_location(C) then
6          discard M;
7      do with probability p;
9          LOC = pseudo_random(rand, this.ID, g);
10         /* LOC is a set of g locations                                  */
11         for all L ∈ LOC do;
12             FM = < ID, L, is_forward, SC > ;
13             this → GPRS (this.LOC, L): FM ;
14             end forall ;
15 else
16     /* forwarding case                                                 */
17     < ID, L, −, SC > ← M ;
18     < C, S > ← SC ;
19     if this.LOC = L then
20         /* this is a witness                                           */
21         if detect_clone(C) then
22             trigger revocation procedure for ID ;
23         put_in_store(C);
24     else
25         n = GPSR(this.LOC, L);
26         if n != this.LOC then
27             this → n : M;
28 end procedure
```

Algorithm 2. RED.

claim is produced by a sensor possessing a secret key coherent with its claimed *ID*. It also verifies that the location contained in the claim is really that of a neighbor. In order to check the signature, each sensor is assumed to be able to fetch the public keys of its neighbors from their *ID*. Based on this control, the neighbor decides either to discard the claim, or to process it. In the latter case, the neighbor, with some probability *p*, puts the claim into a special format, called forwarding, and forwards it to some witnesses. A witness has the task of storing location claims and, whenever it receives a new claim, it checks whether the claim has equal *ID* and different location of another claim already received. In this case it triggers alarm and revokes that *ID*. Since claims are encrypted with secret cryptographic keys of the *ID*, the presence of two such claims is evidence that the *ID* has been compromised. The way in which the witnesses are chosen is the main difference among the (distributed) algorithms that are considered below.

We observe that the fact that each sensor sends its location claim at regular intervals of time is necessary not only for detecting attacks, but also for the normal functioning of the network. In fact, wireless networks are potentially dynamic (because of very different causes, that go from exhaustion of power to compromise and destruction of sensors operated by the attacker) and thus each node has to repeatedly reconstruct the list of its neighbors.

In all the algorithms that we consider in this paper, n is the number of nodes of the network, d denotes the average number of neighbors of each node, p is the probability that a neighbor forwards a location claim, that appears correct, and g is the number of witnesses toward which the claim is forwarded. Additional notation, specific for each algorithm, will be introduced when needed.

2.2. Adversary model

Since the sensors are not tamper-resistant, they can be captured by the attacker that reads the data contained in them and makes copies of them that are called clones. In what follows we will call clone both the first compromised sensor as well as all its copies constructed by the attacker. We assume that the attacker is not able to produce identity-based key pairs that can deceive honest nodes. For instance, because the *good* public keys are certified by a particular certification authority. Hence, all clones of one compromised sensor have same *ID* and cryptographic keys. This assumption is the keystone of the algorithms for detecting clone attacks that we consider below. The attacker can compromise several different nodes and, for each of them, can produce several clones that are added in different positions of the network. However, we always assume that each clone has only one ID and that the total number of nodes controlled by the attacker is much smaller than that of the honest nodes of the network. In fact, there would be no defence against an attack that compromises a large part of the sensors.

3. Existing algorithms for the detection of clone attacks

As explained in the Introduction, our goal is to study how much clone positions influence the functioning of algorithms for detecting clone attacks. Many such algorithms have been proposed in the literature [15,19,21–27]. In [23] most algorithms are presented and classified according to different aspects of their functioning. We decided first to focus on the distributed algorithms, as we think that they are more robust than the centralized ones. Among the distributed algorithms, we have chosen to consider the Line-Selected Multicast (LSM) proposed in [19], the Randomized Efficient and Distributed protocol (RED) proposed in [15], the Single Deterministic Cell (SDC) of [21] and the Random Walk (RAWL) of [22]. In our opinion these algorithms incorporate the most relevant ideas for detecting clone attacks. Some of these algorithms, like RAWL, have recently been extended, for instance in [24], to improve energy and storage consumption. However, these extensions are not relevant for our study which focuses on detection probability.

The rest of this section is devoted to describe the four chosen algorithms in a uniform way, so as to make it easy to understand similarities and differences among them. However, these algorithms are extensions of simpler ones and it is convenient to start the description from these simpler proposals. In the Deterministic Multicast method [19], the witnesses for a given location claim are identified as a function of the ID contained in the claim. Thus, all claims with the same ID are forwarded to the same witnesses which check for claims with the same ID and different location. Unfortunately, this approach has low resiliency against a smart attacker. In fact, also the attacker can compute the witnesses of the *ID* that he has cloned and thus it can compromise all of them in order to avoid detection.

The Randomized Multicast (RM) algorithm [19] is a first attempt to improve the resiliency. In RM the witnesses of each claim are chosen randomly in the network. Thus, two claims with the same *ID* have in general different set of witnesses. Hence, the attack is detected only if the two set of witnesses are not disjoint, but in order to have a reasonable chance that this happens, the sets of witnesses must have high cardinality. If this cardinality is $\sqrt{n}$, then the birthday paradox guarantees a good probability of success. However, forwarding the claims to so many nodes requires many communications and, moreover, in RM nodes need to keep large stores of claims. Thus, in RM the cost of a good detection rate is too high. In Section 3.5, we report the exact figures.

Below we describe four distributed algorithms that improve in different ways on Deterministic Multicast and RM. Before describing the algorithms, we must first introduce the notation that we use for it. This notation allows us to give uniform descriptions of the algorithms which simplifies the understanding of the algorithms and also their comparison.

In all the considered algorithms, each node plays two roles: (i) the role of a sender that broadcasts its location claim to its neighbors and (ii) the role of a receiver that receives either a location claim or a forwarding claim. The first role is realized for all four algorithms by the procedure *Broadcast* given in Section 3.1, whereas the second role is described by four different procedures that are called *ReceiveS*, where *S* specifies the algorithm considered. In all cases, in order to express the fact that a node possesses an identity, a secret key, a position, a set of neighbors and a storage capability, we borrow some notation from object oriented programming as follows. We assume that nodes are objects of some class (for instance of class *sensor*) that has fields *ID, LOC, K, neighbors* and *STORE* and methods *Broadcast*, *ReceiveS*(*M*), *incoherentlocation*(*C*), *detectclone*(*C*), and *putinstore*(*C*). Thus, when executing these methods, the invocation object *this* denotes the node that is executing and *this.ID, this.K, this.LOC, this.neighbors* and *this.STORE* denote, respectively, the identity, the secret key, the location, the set of neighbors, and the store (of location claims) of *this*. The methods *Broadcast* and *ReceiveS*(*M*) correspond to the role of sender and receiver, respectively. The method *incoherentlocation*(*C*), checks that the location contained in *C* is compatible with the local position. The method *detectclone*(*C*) checks *C* against the local store and returns *true* iff the store contains another claim with the same *ID* as *C* and with a different location. In this case, a clone attack is detected. In case that no attack is detected, then *C* is added to the store using *putinstore*. We also use some notation that is rather usual in communication protocols. When we want to express that the current node (*this*) sends a message *M* to another node *n*, we write: *this* → *n: M*. In our procedures we use the symbol "=" for assignments and "←" for pattern matching. We also use some auxiliary functions like *random*(*g*), *pseudorandom*(*rand, ID, g*), *isclaim*(*M*), *badsignature*(*C, S*), and *GPSR*(*LOC, DEST*) whose meaning is as follows.

- *random*(*g*), with $g > 0$ generates *g* random positions.
- *pseudorandom*(*rand, ID, g*) is a function that produces *g* positions, based on a random value *rand*, that is broadcast through the network at regular time intervals, and on the identity *ID* that is being considered.

```
1  Procedure Receive_SDC(M) ;
2  if is_claim(M) then
3      < ID, −, SC > ← M;
4      < C, S > ← SC ;
5      if bad_signature (C, S) || incoherent_location(C) then
6          discard M;
7      do with probability p;
9          Cell = H(this.ID);
10         FM = < ID, Cell, is_forward, SC > ;
11         this → GPRS(this.LOC, Cell): FM ;
12 else
13     /* forwarding case                                          */
14     < ID, Cell, −, SC > ← M ;
15     < C, S > ← SC ;
16     if this.LOC ∈ Cell then
17         /* this is a witness                                   */
18         if detect_clone(C) then
19             trigger revocation procedure for ID ;
20         else
22             do with probability p_s put_in_store(C);
24             foreach L ∈ this.neighbors ∩ Cell do
25                 this → L: M;
26     else
27         n = GPSR(this.LOC, Cell);
28         if n != this.LOC then
29             this → n : M;
30 end procedure
```

Algorithm 3. Receive() for SDC.

```
1  Procedure Receive_RAWL(M) ;
2  if is_claim(M) then
3      < ID, −, SC > ← M;
4      < C, S > ← SC ;
5      if bad_signature (C, S) || incoherent_location(C) then
6          discard M;
7      do with probability p;
8          LOC = random();
9          /* LOC is a set of g random locations                    */
10         for all L ∈ LOC do;
11             FM = < ID, L, is_forward, SC > ;
12             this → GPRS (this.LOC, L): FM ;
13             end forall ;
14 else
15     /* forwarding case                                           */
16     < ID, L, −, SC > ← M ;
17     < C, S > ← SC ;
18     n = GPSR(this.LOC, L);
19     if n != this.LOC then
20         this → n : M;
21     else
23         random_walk(this.LOC, C, 0);
24 end procedure

25 Procedure Random_walk(L, C, i);
26 /* L=current node, C=loc. claim, i=n.of steps done              */
27 if detect_clone(C) then
28     trigger revocation procedure for ID
29 else
30     put_in_store(C);
31     if i + 1 < t then
32         n = random_neighbor();
33         random_walk(n, C, i + 1);
34 end procedure
```

Algorithm 4. Receive() for RAWL.

Table 1
The costs of the four considered algorithms for communications and storage for one single node.

Algorithm	Communication	Storage
LSM	$O(\sqrt{n})$	$O(\sqrt{n})$
RED	$O(\sqrt{n})$	$O(1)$
SDC	$O(\sqrt{n})$	$O(1)$
RAWL	$O(\sqrt{n}\ \log\ n)$	$O(\sqrt{n}\ \log\ n)$

- *isclaim*(*M*) is a boolean function that returns *true* when *M* is a location claim in its original format, that is when it is emitted by a node to prove its position and returns *false* if *M* is a location claim in forwarding format. For simplicity we don't consider that messages can have other forms than these two.
- *badsignature*(*C, S*) returns true iff *S* is the correct signature of *C*.
- when one wants to forward a location claim from *LOC* to *DEST, GPSR*(*LOC, DEST*) returns the next node to which to send the location claim. In case *LOC* coincides with *DEST* or *GPSR* is not able to find a node that has more chance to be connected to *DEST* than *LOC* then *GPSR*(*LOC, DEST*) returns *LOC*. Therefore, a node *this* discovers to be the destination *DEST* (or the closest to *DEST* that GPSR can find) when *GPSR*(*this.LOC, DEST*) returns *this.LOC*. We have called this method *GPSR* to recall that the geographic routing protocol GPSR [32] is used in the simulations described in Section 4.

Finally, let us describe what happens when an attack is detected. A witness discovers a clone attack when it has received two location claims with the same *ID* and different locations. In this case, it floods the network with the two claims and every honest node can check that the attack is real. The soundness of the approach is based on the fact that the two claims contain a signature that proves their authenticity. Thus the attacker cannot construct fake conflicting claims to revoke honest nodes. This operation is denoted as the revocation procedure for the identity *ID*.

3.1. LSM

This section describes the Line-Selected Multicast (LSM) algorithm [19]. The pseudo-code of LSM is given in Algorithm 1. *Broadcast* simply prepares a location claim and sends it to the neighbors of *this*. *Broadcast* will be the same for all four algorithms that are described below.

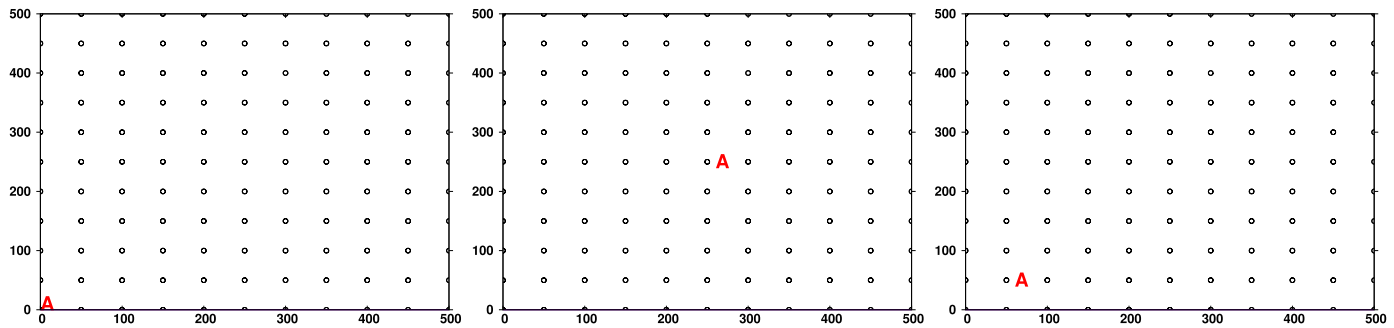
The procedure *ReceiveLSM* specifies the receiver role for LSM. The first part of the procedure, that starts in line 12, describes the case that *isclaim*(*M*) is successful. In this case *this* is the neighbor of a node that has emitted its location claim *M*. After a control of the correctness of the signature and of the coherency of the location contained in the claim with respect to *this.LOC, g* random locations are computed and the claim (in forwarding format) is sent to those destinations. Each neighbor decides to start the forwarding process with probability *p*. The second part of the procedure, that starts in line 24, implements the forwarding phase in which the claim (in forwarding format) traverses a node on its way to the final destinations. The node traversed is a witness and, therefore, it controls that it has not already seen a location claim with the same *ID* and different location. When this happens the node triggers the revocation procedure for the identity *ID*, whereas, if this is not the case, it stores the claim for future checks and forwards it to the next node that is computed with the function *GPSR*. LSM is an improvement on RM in which, instead of scattering the witnesses throughout the network, witnesses are the nodes traversed by some paths. This allows a reasonably good detection rate with a number of witnesses smaller than RM.

3.2. RED

Let us now describe the Randomized Efficient and Distributed protocol (RED), proposed in [15]. A special characteristic of RED is that it assumes that at regular times a random value *rand* is broadcast throughout the network. Thus at each moment of time, all the nodes are supposed to know *rand* and they will use it as explained below. In what follows you find a pseudo-code for RED. It is easy to point out the differences with respect to LSM. *Broadcast* of Algorithm 1 works also for RED. In *ReceiveRED* of Algorithm 2, there are two main differences with respect to *ReceiveLSM*:

- in line 9, the locations of the witnesses are computed using function *pseudorandom* which takes as input the current value *rand*, the *ID* of the claim, the integer $g \geq 1$ and produces a set *LOC* of *g* witnesses to which the location claim must be forwarded. Observe that, since all nodes use the same *rand*, two location claims with the same *ID* (and different locations) will be sent to the same witnesses, which will surely discover the attack.
- the nodes traversed on the way to the witnesses do not make any control of clone repetition, they simply forward the claims toward the witnesses. Only witnesses control a claim they receive against their store.

It should be clear that RED is, for several aspects, very similar to Deterministic Multicast, that we have seen in the initial part of this section. However, RED greatly improves the weakness of Deterministic Multicast with respect to smart attacks because it computes the witnesses using the *rand* value which changes at regular time intervals. Thus, the witnesses corresponding to a given *ID* are different in different intervals of time and therefore, even a smart attacker could hardly protect its clones from detection for several intervals of time.

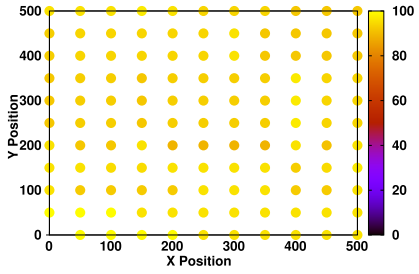


(a) A Clone is fixed in $(0, 0)$ (b) A Clone is fixed in $(250, 250)$ (c) A Clone is fixed in $(50, 50)$

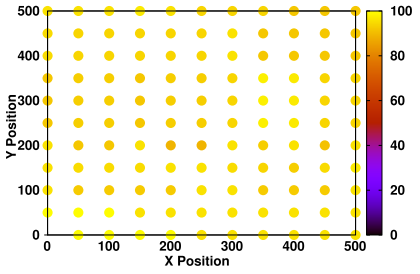
Fig. 1. Square topology.

3.3. Localized multicast

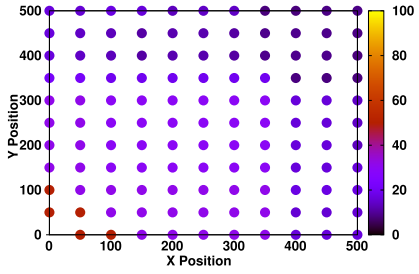
In [21] a new approach is put forward for the distributed detection of clone attacks, the Localized Multicast. The approach is based on dividing the network in cells and sending the location claims to one or several cells on the basis of the *ID* of the claim. This seems dangerous because the cell(s) can also be computed by the attacker, but resiliency is restored by having a sufficiently large number of nodes in each cell and by flooding the claim to all these nodes and having each of them decide with probability *g* whether it stores the claim or not (i.e., if it becomes a witness for the claim or not). Thus the attacker either subverts all nodes in the cell or otherwise has a very small chance to control all witnesses. This approach is inspired again by Deterministic Multicast: the cell (or cells) is determined by the *ID*, but the random of RM is recovered by choosing randomly the witnesses inside that cell(s). In [21] two protocols are presented that follow this approach: the



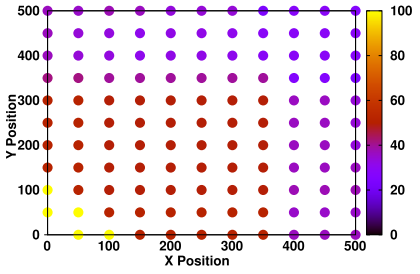
(a) RED: 500 Nodes



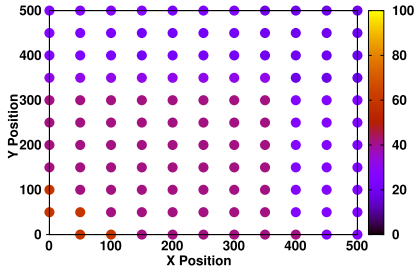
(e) RED: 5000 Nodes



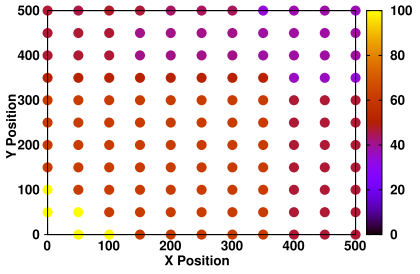
(b) LSM: 500 Nodes



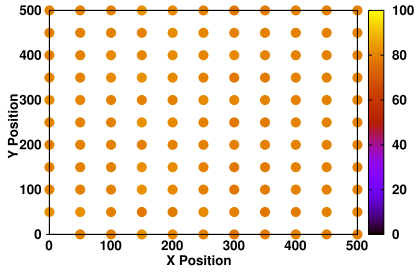
(f) LSM: 5000 Nodes



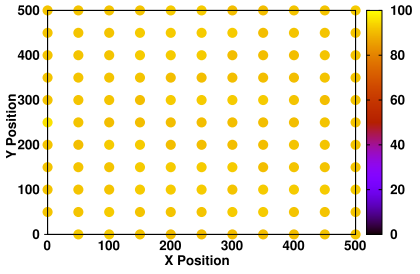
(c) RAWL: 500 Nodes



(g) RAWL: 5000 Nodes

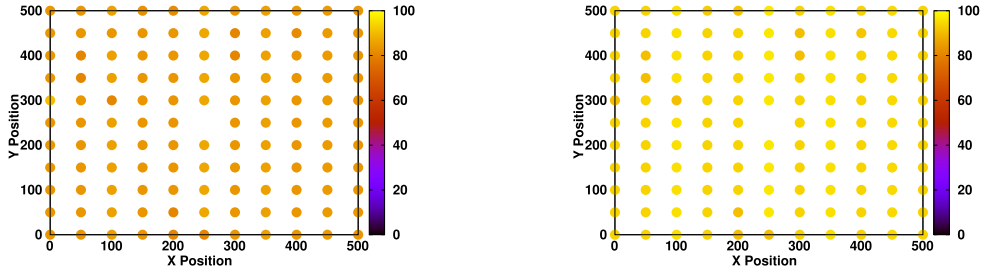


(d) SDC: 500 Nodes



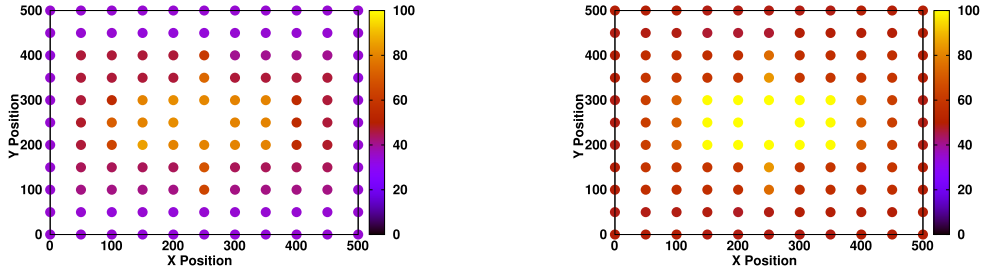
(h) SDC: 5000 Nodes

Fig. 2. Square topology: fixed clone at position (0, 0).



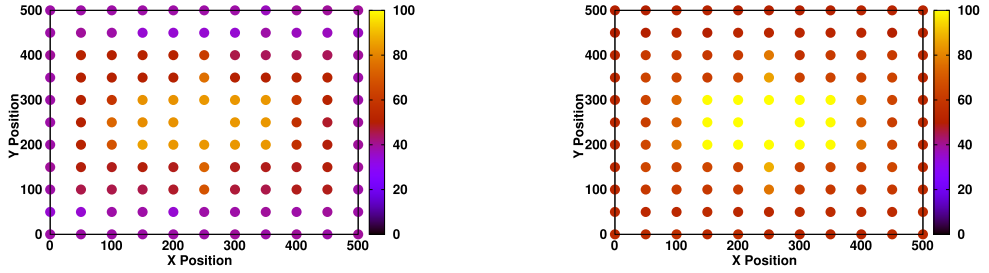
(a) RED: 500 Nodes

(e) RED: 5000 Nodes



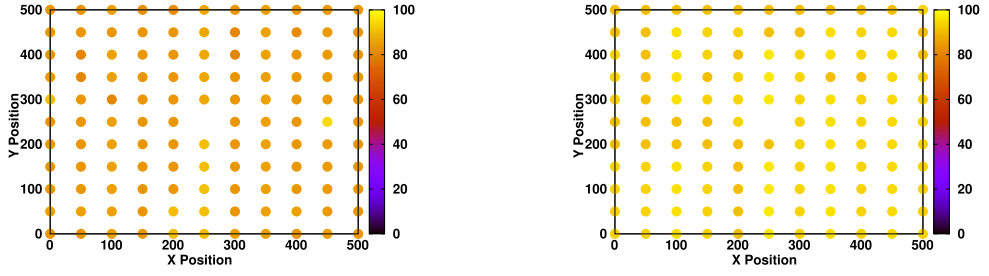
(b) LSM: 500 Nodes

(f) LSM: 5000 Nodes



(c) RAWL: 500 Nodes

(g) RAWL: 5000 Nodes



(d) SDC: 500 Nodes

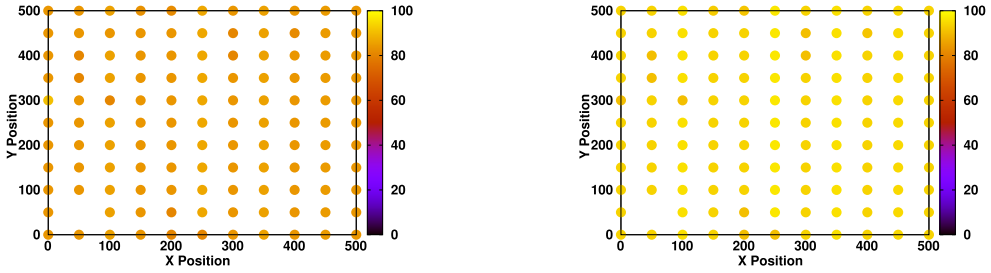
(h) SDC: 5000 Nodes

Fig. 3. Square topology: fixed clone at position (250, 250).

Single Deterministic Cell (SDC) and the Parallel Multiple Probabilistic Cell (P-MPC). SDC follows the idea described before: location claims are sent to one cell and flooded to all nodes of the cell some of which *decide* to be witnesses. P-MPC sends the location claims to several cells with different probabilities. This makes it more resilient even against attackers ready to subvert a number of nodes comparable to those of a single cell. In what follows, we present the pseudo-code for SDC. We will not consider P-MPC.

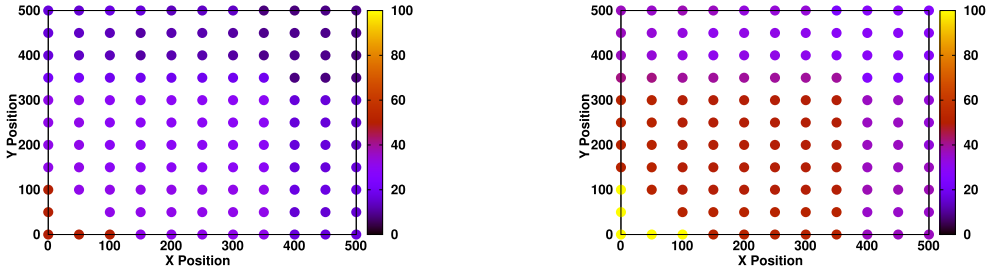
The procedure *Broadcast* of Algorithm 1 works also for SDC. *ReceiveSDC*, given in Algorithm 3, is obviously very similar to the previous *Receive* procedures. The differences are as follows:

- In line 9 of *ReceiveSDC*, a neighbor computes a cell and forwards the claim toward that cell. This clearly assumes that each node has a knowledge of the number and positions of the cells that compose the deployment field. Observe also that we use *GPSR* with a cell as second parameter, instead of a location as done so far. The idea is that *GPSR* should compute the next step to reach the cell.



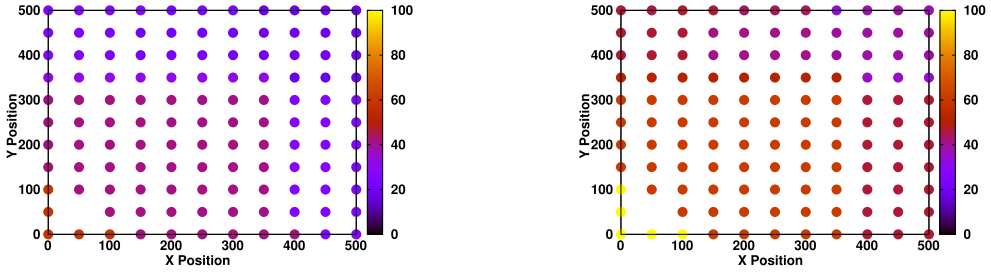
(a) RED: 500 Nodes

(e) RED: 5000 Nodes



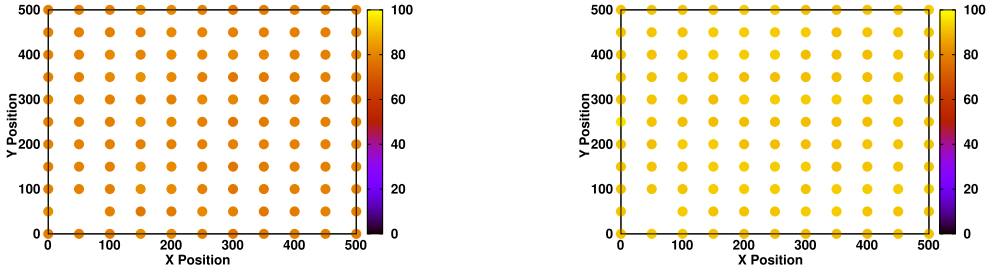
(b) LSM: 500 Nodes

(f) LSM: 5000 Nodes



(c) RAWL: 500 Nodes

(g) RAWL: 5000 Nodes



(d) SDC: 500 Nodes

(h) SDC: 5000 Nodes

Fig. 4. Square topology: Scenario #3, One clone at position (50, 50).

- In lines 22 and 24, we model the operation of flooding the location claim through the cell and also the decision of each node of the cell whether to become a witness. We notice that the check against the store is done by each node of the cell independently of whether it becomes a witness.

3.4. Random walk

In [22], the authors propose the random walk strategy approach for detecting repetition attacks. The idea is simple: each neighbor, with probability p sends the location claim to g random locations and, once the claim reaches any of these locations, the algorithm follows a random walk of k hops. Each node traversed by such a walk is a witness and stores the claim. An attack is detected when two random walks

generated by two location claims with the same *ID* intersect each other. The intersection node has both claims and thus it can detect the repetition attack. In [22], the authors propose two algorithms based on this approach: the Random Walk (RAWL) and the Table-assisted Random Walk (TRAWL). In what follows we only consider RAWL.

TRAWL is a variation of RAWL meant to improve the space requirement of RAWL. As for the preceding algorithms, the *Broadcast* procedure of Algorithm 1 works also for RAWL. *ReceiveRAWL* procedure in Algorithm 4 contains the procedure *randomwalk* that is used by *ReceiveRAWL*. An important point of *ReceiveRAWL* is line 23, in which a node is reached such that a random walk starts from it. This is done by the new procedure *randomwalk* which executes a random walk through the network for t steps. The choice of the parameter t is a compromise between precision and efficiency. In fact, a large value of t assures higher detection of attacks, but also causes higher communication and storage costs. Observe that the procedure *randomwalk* uses *randomneighbor*() to find the next node to go in the random walk. Clearly RAWL takes some inspiration from LSM that uses the idea that it enhances detection to have witnesses along paths because paths can easily cross each other in a sensor that would then receive the two contradictory location claims. However, in LSM all paths start in the proximity of the node that emitted the claim and this may help a smart attacker. To avoid this, RAWL starts random walks in random points of the network, even very distant from the start of the process.

3.5. Costs of the algorithms

In this section, we recall the different costs of the four representative algorithms in [15,19,21,22]. The analysis in terms of communication and storage in Table 1 can be found in the corresponding articles [15,19,21,22]. We mention that n is the number of nodes of the network. As mentioned by authors in [22], they consider the asymptotic number of messages to analyse the protocols behavior.

4. Simulation results

We evaluated the performances of the four algorithms RED [15], LSM [19], SDC [21], and RAWL [22] using the VISIDIA simulator [28,29], that is a java based tool that deals with events, and has been used for implementing, simulating, testing and showing the correctness of distributed algorithms [30]. We used VISIDIA in order to evaluate the probability that the four algorithms detect a clone attack represented by the presence of two clones. The aim of this simulation is to find out how much the detection probability changes with the position of the two clones. This is done for different scenarios in which the WSN is deployed. In Section 4.1, we describe the parameters that are used for the simulation. The successive parts of Section 4 describe the different scenarios that are considered and, for each scenario, describe the corresponding results of the simulation. The scenarios that are considered are of two types: on the one hand we consider a square playground in which the sensors composing the WSN are uniformly distributed and, on the other hand, we consider scenarios in which, inside the playground, there are large areas (that we call holes) that have no sensor, whereas sensors are uniformly distributed on the rest of the playground. WSN with holes are interesting because they are more realistic models of real deployments on natural areas. When holes are present, the behavior of the algorithms becomes strongly dependent on the positions of the two clones, for instance our study shows that there are particular positions for the clones that make their detection difficult for all algorithms.

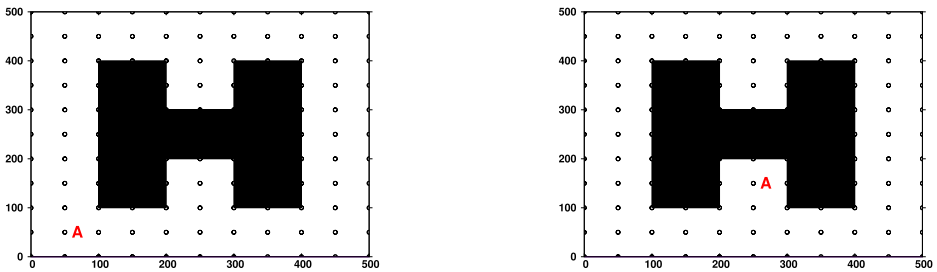
4.1. Simulation parameters

The playground of the WSN has always dimension of 500 × 500 units and sensors are randomly distributed. For each deployment, we considered 500 simulations of which we take the average. The transmission range of the sensors varies in such a way that the average number of neighbors for each node is 40. In order to evaluate the importance of the WSN density, the total number of sensors is either 500 or 5000. In all cases, routing is done by the geographic routing protocol GPSR as defined in [32] (including the right hand rule). The number of witnesses g is always fixed to 1. In RAWL protocol, the length of the random walk is 10 as in [22]. For SDC, as in the original paper, [21], we consider that the playground is divided in 9 × 9 cells. In order to obtain results as independent as possible of the particular positions of the sensors, we consider, for each situation, 10 different random deployment of the sensors, i.e., 10 WSN. The figures that are given below should be read as follows. In each figure there is one clone that is put in a fixed position, whereas the second clone varies in all other positions. For instance, in Fig. 1(a) the fixed clone is in position (0, 0) whereas the second clone varies in all other points represented in the figure. For each position of the second clone, we consider 10 different random deployments, and for each deployment the simulation indicates whether the clone attack is detected. In this scenario, 10 detections give the maximum probability of detection (100%) that corresponds to a point in that position of the brightest color, whereas 0 detections is the worse case (0%) that corresponds to a black dot in that point.

4.2. Square topology

The first scenario that we consider is that the WSN is uniformly distributed on a square playground. In this scenario, we consider that one clone is in one of the following three positions: (0, 0), (250, 250) and (50, 50). Fig. 1(a)–(c) show these three positions on the square playground. All other points in the pictures are the positions in which the second clone could be.

Fig. 2 shows the results of the simulation when a clone has a fixed position in (0, 0). The left column shows the results for the case that the WSN consists of 500 sensors, whereas the right column represents the case with 5000 sensors. Each column has 4 parts, one for each of the 4



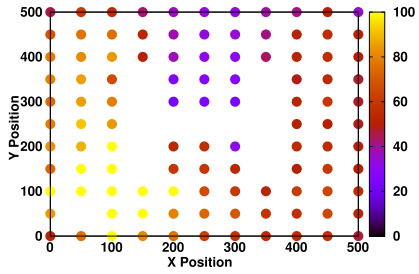
(a) Fixed Clone at Position (50, 50)

(b) Fixed Clone at Position (250, 150)

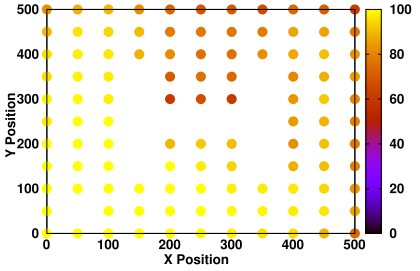
Fig. 5. H Topology.

algorithms that we are considering. The behavior of RED is shown in Fig. 2(a). It is easy to see that RED is practically insensitive to the position of the second clone. The probability of detection is slightly higher only when the second clone is in the immediate neighborhood of the first one. For RED, the average detection probability is more than 70% in the case that the WSN are composed of 500 sensors, whereas, as expectable, the probability grows to about 87% when the number of sensors is 5000. Also in this case the detection probability is practically insensitive to the position of the second clone, with a slight improvement in detection in the immediate neighborhood of (0, 0).

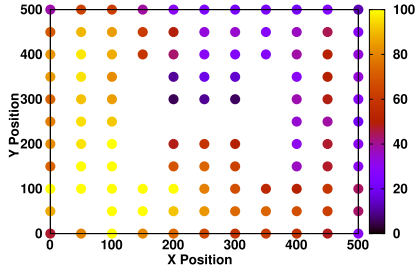
Fig. 2(b) and (f) present the behavior of LSM with the WSN composed of 500 and 5000 sensors. As expected, with 5000 sensors, the detection probability increases, but in both cases the detection probability appears to be strongly dependent of the position of the second clone. In fact, in both figures, three fairly uniform regions can be distinguished: when the second clone is within 200 units distance from (0, 0) the detection probability is more than 60%, when the second clone is at distance between 200 units and 400 units from (0, 0) then the



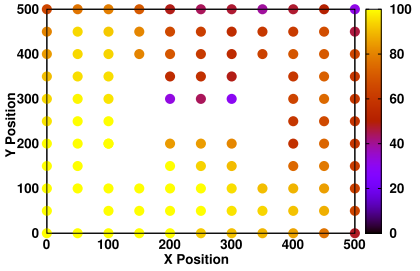
(a) RED: 500 Nodes



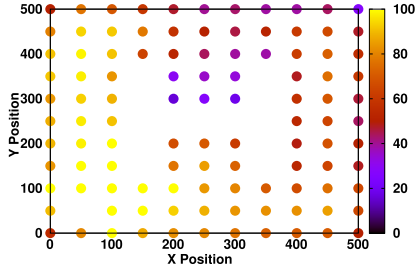
(e) RED: 5000 Nodes



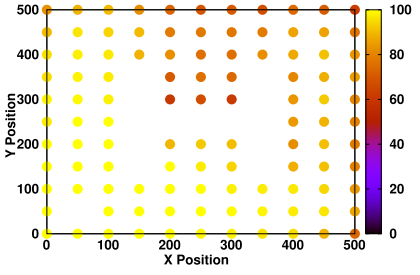
(b) LSM: 500 Nodes



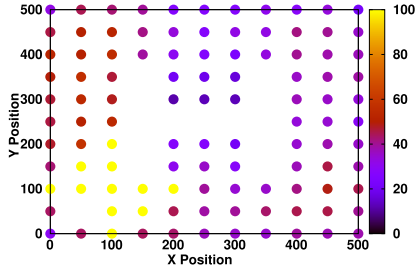
(f) LSM: 5000 Nodes



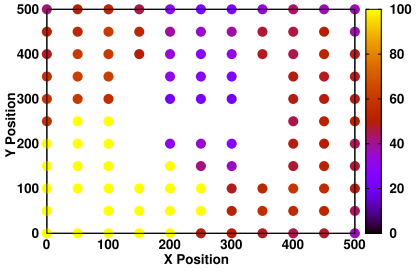
(c) RAWL: 500 Nodes



(g) RAWL: 5000 Nodes

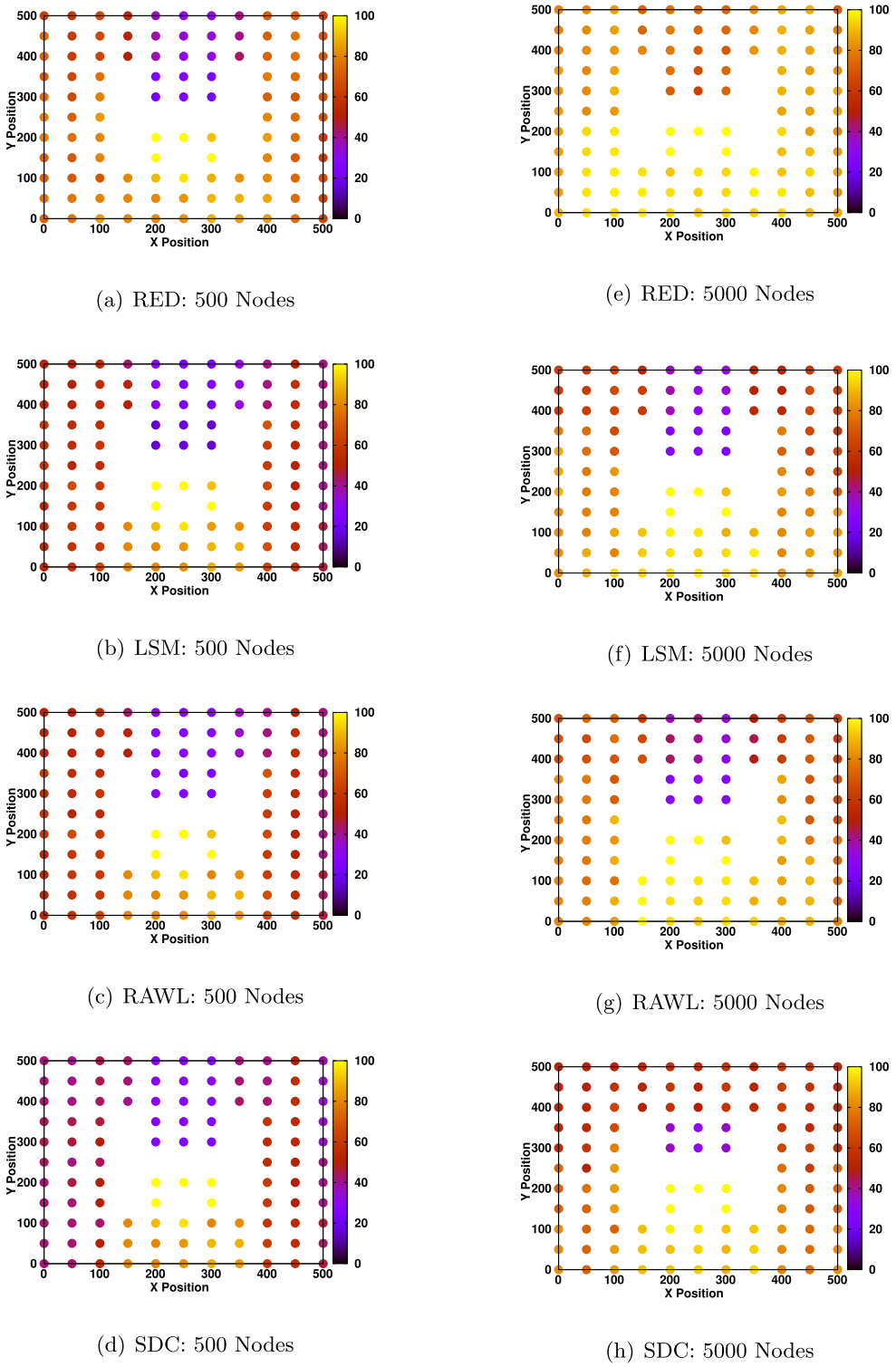


(d) SDC: 500 Nodes



(h) SDC: 5000 Nodes

Fig. 6. H thin topology: one clone at position (50, 50).



(a) RED: 500 Nodes

(e) RED: 5000 Nodes

(b) LSM: 500 Nodes

(f) LSM: 5000 Nodes

(c) RAWL: 500 Nodes

(g) RAWL: 5000 Nodes

(d) SDC: 500 Nodes

(h) SDC: 5000 Nodes

Fig. 7. H Thin Topology: one clone at position (250, 150).

detection probability varies between 30% and 40%, and finally when the second clone is at a distance bigger than 400 units from (0, 0) then the detection probability drops to less than 20%. This could be explained by the fact that the two paths originating in the neighborhood of the two clone nodes have a higher probability to intersect when the two clones are near.

Fig. 2(c) and (g) show the simulation of RAWL. It is interesting to note that RAWL behaves similarly to LSM. In fact, RAWL is influenced by the distance of the second clone from (0, 0). One can distinguish also for RAWL the presence of three regions of decreasing detection probabilty as for LSM. However, compared to LSM, the detection probability of RAWL increases slightly for all the positions of the second clone.

Finally, Fig. 2(d) and (h) show the behavior of SDC protocol with 500 and 5000 sensors. SDC has a behavior similar to RED. It is mainly insensitive to the position of the second clone. One observes however that SDC has, in general, a lower detection probability compared to RED.

Figs. 3 and 4 consider the fixed clone, respectively, in position (250, 250) and (50, 50). They show behaviors similar to the ones shown in Fig. 2. RED and SDC have detection probability rather insensitive with respect to the second clone position, whereas LSM and RAWL have detection probability that clearly decreases with the distance of the second clone with respect to the fixed clone. As for Fig. 2, one observes, for LSM and RAWL, three different areas with rather uniform detection probability that decreases with the distance from the fixed clone. When the fixed clone is in position (250, 250), this phenomenon is more evident with 500 sensors.

4.3. The H topology

We have also studied the behavior of RED, LSM, SDC, and RAWL with the scenarios shown in Fig. 5. The black H is an area of the playground that has no sensors. For this reason we call it a hole. Fig. 5 shows the two positions of the fixed clone that are considered for this scenario: (50, 50) and (250, 150). It is important to understand that sensors that are on different sides of the H cannot communicate with each other directly. Moreover, the sensors are not *aware* of the position of the H on the playground. Thus, a witness or even the witness cell in SDC, may fall on top of the H and thus they may be difficult or even impossible to reach. Such situations can deteriorate the detection probability of the algorithms.

Figs. 6 and 7 show the result of the simulations for the two scenarios depicted in Fig. 5.

When we contrast these Figures with those of the square topology, we observe the following interesting phenomena:

- The H-shaped hole has two niches that are good hiding places for clones. All four algorithms have their lowest detection probability when one clone is in one such niche and the other clone is on the other side of the H. This probability is different for the different algorithms, with RED showing the highest probability (around 50%), whereas the other three algorithms have a probability around 30%.
- Each algorithm behaves similarly with 500 and with 5000 sensors. However, with 5000 sensors the detection probability is generally higher than with 500.
- RED and SDC are sensitive to the relative positions of the two clones. Moreover, their detection probability is generally lower than what they show in the uniform square scenario. The highest probability (similar to the uniform square case) is attained when the two clones are close to each other. A plausible reason for this is as follows. In RED and SDC, the location claims originating from the neighbors of the two clones must reach the same witness (or the same cell for SDC). Thus, if the two clones are close, the chance that both claims reach the witness (the cell for SDC) is big. On the contrary, when the clones are far apart, then it may happen that one claim may reach the witness (cell for SDC) whereas the other claim does not. Thus the attack is not detected. In general, the presence of the hole makes more difficult for location claims to reach the witness (witness cell). This explains the decrease in performance of these algorithms in this scenario.
- LSM and RAWL show a striking improvement with respect to the uniform square scenario. They can compete now with RED and SDC. The reason is probably that LSM and RAWL, take advantage of the presence of a H-shaped hole in the playground. In fact, in such a scenario, all the random paths must go around the hole and this increases the probability that the paths cross each other. LSM remains less precise than RAWL, but its improvement with respect to the uniform square scenario is very clear.

Table 2
Detection probability for the cross cases (%).

Algorithm	Case 1	Case 2	Case 3
LSM	31.2	33.8	54.6
RED	53.6	55.2	78.2
SDC	29.8	27.2	47.8
RAWL	38.2	36.6	56.2

In order to study further the effect of niches as hiding places for clones, we have considered a playground with a hole in the form of a cross, cf. Fig. 8 with clones in the following positions:

1. Case 1: two clones in positions (100, 400) and (400, 100);
2. Case 2: two clones in positions (150, 350) and (350, 150);
3. Case 3: four clones in positions (150, 350), (350, 350), (150, 150) and (350, 150).

The detection probability of the four algorithms for these three cases is shown in the Table 2.

Case 3 in which the four clones are in adjacent niches shows for all algorithms the best detection probability. The figures for Cases 1 and 2 show that non adjacent niches are good hiding places for clones. These observations indicate that in order to improve clone detection, one should eliminate the effect of niches, for instance, by isolating sensors that happen to be in a niche.

RED has good detection probability when clones are in adjacent niches, whereas the other three algorithms have mediocre performances, but still much better than when the niches are not adjacent.

4.4. Four round holes topology

After having studied the effect of niches as good hiding places for

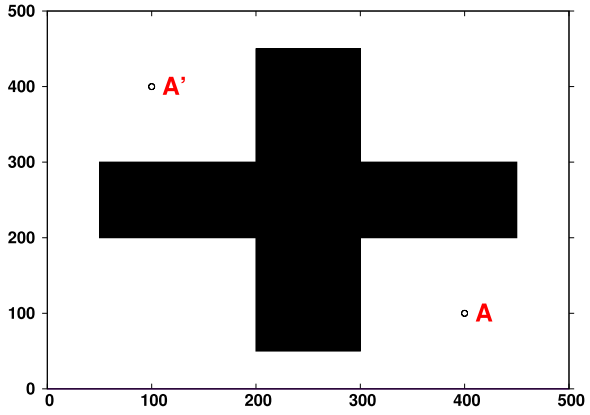


Fig. 8. Cross topology, Case 1 with two clones in positions (100, 400) and (400, 100).

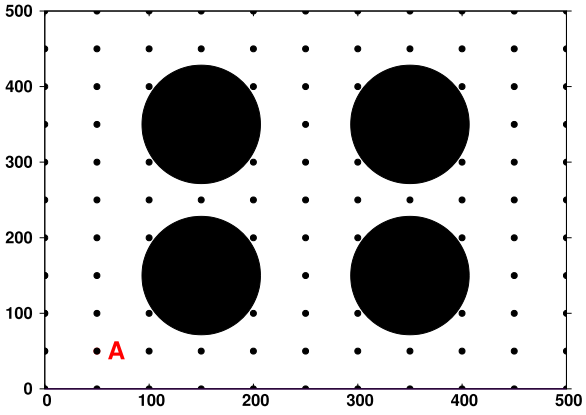
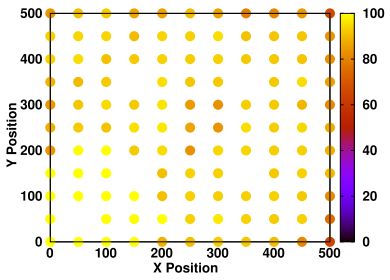
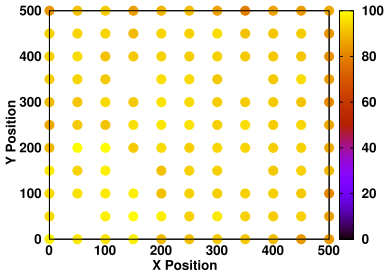


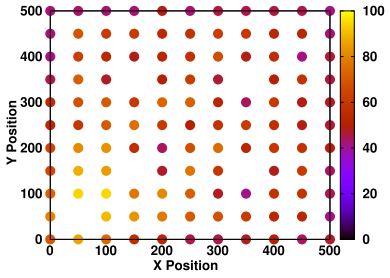
Fig. 9. Square with 4 round holes.



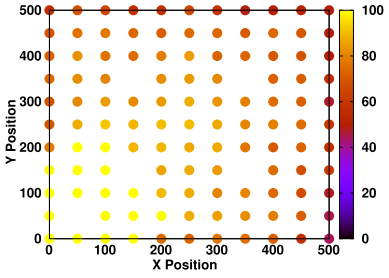
(a) RED: 500 Nodes



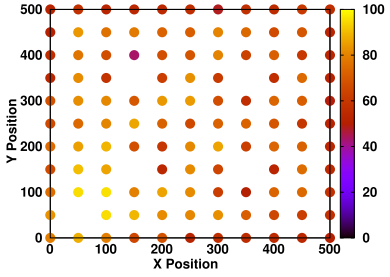
(e) RED: 5000 Nodes



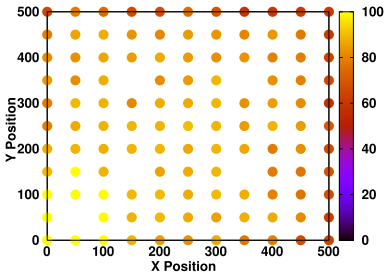
(b) LSM: 500 Nodes



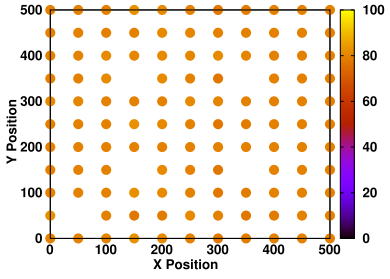
(f) LSM: 5000 Nodes



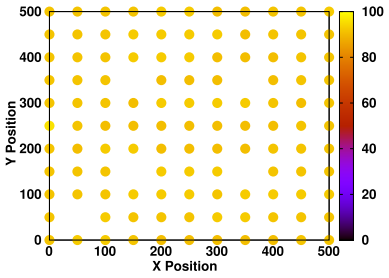
(c) RAWL: 500 Nodes



(g) RAWL: 5000 Nodes



(d) SDC: 500 Nodes



(h) SDC: 5000 Nodes

Fig. 10. Four round holes topology: One Clone at position (50, 50).

clones, we also want to consider the effect of holes without niches on the performance of the four algorithms. To this end, we considered the scenario depicted in Fig. 9.

Fig. 10 shows the results of the simulation in this scenario. We observe several interesting phenomena:

- RED and SDC present a fairly uniform probability, rather similar to the uniform square scenario. Thus, it seems that smaller holes without hiding places are less harmful for these algorithms than one big central hole. Fig. 10 shows that when clones are far apart their location claims reach the witness (witness cell) with high probability. This was not the case for the H-shaped hole.
- LSM and RAWL show a behavior similar to that of Figure 6, but the probability values are generally lower than those of Fig. 6. The four round holes determine some forced directions for random walks, but they leave more choices than the H-shaped hole and thus random walks may take different directions. LSM does perform particularly well when the two clones are very close.

5. Conclusion

In this article we measure the impact that the positioning of clones has on the probability of detecting their presence. As far as we know, it is the first time that such a study is carried out. After a thorough study of the literature, we have chosen to focus our study on the following four distributed algorithms for clone detection: LSM, RED, SDC and RAWL. These algorithms are described in a uniform and precise way in Section 3. We have used the VISIDIA simulator for modelling these four algorithms in order to measure the performance of these algorithms when they are used with different configurations of WSN and with different positions of two clones. The main conclusions of this simulation follow.

With networks in which sensors are uniformly distributed, RED and SDC are not affected by the clone positions. On the contrary the detection probability of LSM and RAWL worsens with the distance of the two clones. This was somehow expectable because of the different ways in which RED and SDC on the one side and LSM and RAWL on the other side, choose the witnesses, cf. Section 3. Less expectable is what happens when configurations with holes (i.e., areas without sensors) are considered. With such configurations all four algorithms are affected by the clone positions. When holes have a form that contains niches then clearly these niches become positions in which clones are difficult to detect, independently of the algorithm that is used. Even more surprisingly, when holes are present, LSM and RAWL improve their detection probability with respect to the uniform case. We observe that this improvement is more evident when the holes are such that the random paths of LSM and RAWL have only few directions in which they can grow.

These observations suggest that there is no algorithm that is best in all circumstances. On the contrary, the particular configuration of a wireless network is relevant when choosing an appropriate clone detection algorithm for that network. It is also worthwhile to observe that the strong effect that holes have on the performance of algorithms, suggests an interesting technique that may be used to improve clone detection together with power consumption within a network. Areas of the network may simply turn off their communications for a while, creating an artificial hole for some time. As our observations indicate, this may improve clone detection, but it may also improve power efficiency because in this way communications can be better distributed among all sensors.

Acknowledgments

This work is partially supported by the EU project NRG-5 (agreement H2020-ICT-2016-2-762013) and the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061).

References

[1] K.B. Rasmussen, S. Capkun, Implications of radio fingerprinting on the security of sensor networks, International Conference on Security and Privacy in Communications Networks SecureComm, (2007), pp. 331–340.
[2] A. Becher, Z. Benenson, M. Dornseif, Tampering with motes: Real-world physical attacks on wireless sensor networks, International Conference on Security in Pervasive Computing, (2006), pp. 104–118.
[3] K. Xing, F. Liu, X. Cheng, D.H. Du, Real-time detection of clone attacks in wireless sensor networks, The 28th International Conference on Distributed Computing Systems ICDCS, (2008), pp. 3–10.
[4] H. Choi, S. Zhu, T.F.L. Porta, Set: Detecting node clones in sensor networks, Third International Conference on Security and Privacy in Communications Networks, SecureComm, (2007), pp. 341–350.
[5] M. Conti, R.D. Pietro, L. Mancini, A. Mei, Distributed detection of replica node attacks with group deployment knowledge in wireless sensor networks, Ad Hoc Netw. 7 (8) (2009) 1476–1488.
[6] W.T. Zhu, J. Zhou, R.H. Deng, F. Bao, Detecting node replication attacks in wireless sensor networks: a survey, J. Netw. Comput. Appl. 35 (3) (2012) 1022–1034.
[7] R. Di Pietro, S. Guarino, N.V. Verde, J. Domingo-Ferrer, Security in wireless ad-hoc networks–a survey, Comput. Commun. 51 (2014) 1–20.
[8] C.-L. Fok, G.-C. Roman, C. Lu, Rapid development and flexible deployment of adaptive wireless sensor network applications, 25th IEEE International Conference on Distributed Computing Systems ICDCS, (2005), pp. 653–662.
[9] W. Ben Jaballah, M. Conti, R. Di Pietro, M. Mosbah, N.V. Verde, Mass: An efficient and secure broadcast authentication scheme for resource constrained devices, 2013 IEEE CRiSIS, (2013), pp. 1–6.
[10] Z. Li, G. Gong, On the node clone detection in wireless sensor networks, IEEE/ACM Trans. Netw. 21 (6) (2013) 1799–1811.
[11] C.-M. Yu, C.-S. Lu, S.-Y. Kuo, Csi: compressed sensing-based clone identification in sensor networks, IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), (2012), pp. 290–295.
[12] D.-J. Huang, W.-C. Teng, A defense against clock skew replication attacks in wireless sensor networks, J. Netw. Comput. Appl. 39 (2014) 26–37.
[13] Y. Zhou, Y. Fang, Y. Zhang, Securing wireless sensor networks: a survey, IEEE Commun. Surveys Tutorials 10 (3) (2008) 6–28.
[14] R. Di Pietro, C. Soriente, A. Spognardi, G. Tsudik, Collaborative authentication in unattended wsns, Proceedings of the Second ACM Conference on Wireless Network Security WISEC, (2009), pp. 237–244.
[15] M. Conti, R.D. Pietro, L. Mancini, A. Mei, Distributed detection of clone attacks in wireless sensor networks, IEEE Trans. Dependable Secure Comput. 8 (5) (2011) 685–698.
[16] M. Conti, Secure Wireless Sensor Networks, vol. 65, Springer-Verlag, 2015.
[17] M. Conti, R. Di Pietro, A. Spognardi, Clone wars: distributed detection of clone attacks in mobile wsns, J. Comput. Syst. Sci. 80 (3) (2014) 654–669.
[18] M. Conti, R. Di Pietro, L.V. Mancini, A. Mei, A randomized, efficient, and distributed protocol for the detection of node replication attacks in wireless sensor networks, Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing MobiHoc, (2007), pp. 80–89.
[19] B. Parno, A. Perrig, V. Gligor, Distributed detection of node replication attacks in sensor networks, IEEE Symposium on Security and Privacy, (2005), pp. 49–63.
[20] M. Zhang, V. Khanapure, S. Chen, X. Xiao, Memory efficient protocols for detecting node replication attacks in wireless sensor networks, 17th IEEE International Conference on Network Protocols ICNP, (2009), pp. 284–293.
[21] B. Zhu, S. Setia, S. Jajodia, S. Roy, L. Wang, Localized multicast: efficient and distributed replica detection in large-scale sensor networks, IEEE Trans. Mob. Comput. 9 (7) (2010) 913–926.
[22] Y. Zeng, J. Cao, S. Zhang, S. Guo, L. Xie, Random-walk based approach to detect clone attacks in wireless sensor networks, IEEE J. Sel. Areas Commun. 28 (5) (2010) 677–691.
[23] A.K. Mishra, A.K. Turuk, A comparative analysis of node replica detection schemes in wireless sensor networks, J. Netw. Comput. Appl. 61 (2016) 21–32.
[24] M. Dong, K. Ota, L.T. Yang, A. Liu, M. Guo, Lscd: a low-storage clone detection protocol for cyber-physical systems, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 35 (5) (2016) 712–723.
[25] Z. Zheng, A. Liu, L.X. Cai, Z. Chen, X. Shen, Energy and memory efficient clone detection in wireless sensor networks, IEEE Trans. Mob. Comput. 15 (5) (2016) 1130–1143.
[26] W.Z. Khan, M.S. Hossain, M.Y. Aalsalem, N. Saad, M. Atiquzzaman, A cost analysis framework for claimer reporter witness based clone detection schemes in wsns, J. Netw. Comput. Appl. 63 (C) (2016) 68–85.
[27] L.B.R.P. T. Bonaci P. Lee, A convex optimization approach for clone detection in wireless sensor networks, Elsevier Pervasive and Mobile Computing 9 (4) (2013) 528–545.
[28] https://visidia.labri.fr.
[29] W. Abdou, N.O. Abdallah, M. Mosbah, Visidia: A java framework for designing, simulating, and visualizing distributed algorithms, 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications, (2014), pp. 43–46.
[30] T. Morsellino, C. Aguerre, M. Mosbah, Debugging the execution of distributed algorithms over anonymous networks, 16th International Conference on Information Visualisation, (2012), pp. 464–470.
[31] F. Hess, Efficient identity based signature schemes based on pairings, SAC, (2002).
[32] B. Karp, H.T. Kung, Gpsr: Greedy perimeter stateless routing for wireless networks, Proceedings of the 6th Annual International Conference on Mobile Computing and Networking MobiCom, (2000), pp. 243–254.