J. Sens. Actuator Netw. **2015**, *4*, 378-409; doi:10.3390/jsan4040378

OPEN ACCESS

Journal of Sensor and Actuator Networks
ISSN 2224-2708
www.mdpi.com/journal/jsan

Article

The Efficacy of Epidemic Algorithms on Detecting Node Replicas in Wireless Sensor Networks [†]

Narasimha Shashidhar [1,*], Chadi Kari [2,‡] and Rakesh Verma [3,‡]

[1] Department of Computer Science, Sam Houston State University, Huntsville, TX 77341, USA; E-Mail: karpoor@shsu.edu

[2] Department of Computer Science, University of The Pacific, Stockton, CA 95211, USA; E-Mail: celkari@pacific.edu

[3] Department of Computer Science, University of Houston, Houston, TX 77004, USA; E-Mail: rmverma@cs.uh.edu

[†] This paper is an extended version of our paper published in "Shashidhar, N.; Kari, C.; Verma, R. Epidemic Node Replica Detection in Sensor Networks. In Proceedings of the Third IEEE ASE International Conference on Cyber Security (CyberSecurity 2014), Stanford, CA, USA, 27–31 May 2014".

[‡] These authors contributed equally to this work.

* Author to whom correspondence should be addressed; E-Mail: karpoor@shsu.edu; Tel.: +936-294-1591; Fax: +936-294-4312.

Academic Editor: Dharma P. Agrawal

Received: 15 October 2015 / Accepted: 9 December 2015 / Published: 14 December 2015

Abstract: A node replication attack against a wireless sensor network involves surreptitious efforts by an adversary to insert duplicate sensor nodes into the network while avoiding detection. Due to the lack of tamper-resistant hardware and the low cost of sensor nodes, launching replication attacks takes little effort to carry out. Naturally, detecting these replica nodes is a very important task and has been studied extensively. In this paper, we propose a novel distributed, randomized sensor duplicate detection algorithm called Discard to detect node replicas in group-deployed wireless sensor networks. Our protocol is an epidemic, self-organizing duplicate detection scheme, which exhibits emergent properties. Epidemic schemes have found diverse applications in distributed computing: load balancing, topology management, audio and video streaming, computing aggregate functions, failure detection, network and resource monitoring, to name a few. To the best of our knowledge, our algorithm is the first attempt at exploring the potential of this paradigm to detect replicas in a wireless sensor network. Through analysis and simulation, we show that our scheme achieves robust replica detection with substantially lower communication, computational and storage requirements than prior schemes in the literature.

Keywords: replica detection; duplicate detection; wireless sensor networks; emergent algorithms; epidemic gossip

---

1. Introduction

A sensor network is composed of a number of low-cost nodes typically scattered in a sensor field. The purpose of these nodes is to sense the parameters of interest and report their findings, either to the base station or to share this information with their neighbors. Nodes join the network by initiating simple neighbor discover protocols [1–4] without requiring any registration procedure with a base station. Sensor nodes, by design, are low-cost devices and, to this end, are built and deployed without any tamper-resistant features. This implies that an adversary can obtain access to the contents of a sensor node and subsequently duplicate the node and deploy the replicas back in the network with reasonable ease. If left undetected, such duplicate nodes can defeat or subvert the goals of the network. For instance, with a sufficient number of duplicates in the network, an adversary can inject false data into the network, suppress legitimate data, partition or disconnect the network and subsequently launch other malicious attacks. Clearly, detection of these replicated nodes in the network is of paramount importance and has garnered significant interest from the research community. In this paper, we study the problem of detecting these replica nodes in a group-deployed, two-dimensional, stationary wireless sensor network and present a duplicate detection scheme, which respects the constraints imposed by the sensor nodes and the wireless sensor network. Hence, in our protocol, we aim to minimize the communication, storage and computation costs incurred by individual nodes and the network as a whole, while achieving robust detection and security against the adversary, whose objective is to thwart detection.

One of our goals in this paper is to study the fitness of epidemic algorithms in detecting replica nodes in a sensor network. To this end, we propose a novel distributed, randomized sensor duplicate detection algorithm called **Discard** to detect node replicas in group-deployed wireless sensor networks. Our scheme is built using an epidemic message propagation paradigm. We show that our epidemic-gossip-based scheme exhibits emergent properties, *i.e.*, characteristic properties that are manifested only as a consequence of interactions among members of a group and absent otherwise. We designed **Discard** in a very organic manner: our idea was to design a protocol that emulates the natural workings of a sensor network. As noted above, sensor nodes communicate with each other periodically, sending and receiving heart-beat messages, membership messages, neighbor discovery messages and the sensed data from the environment. Our scheme is designed in harmony with the natural processes already at play in the network. This ensures that **Discard** can be integrated into a sensor network in a seamless fashion.

Epidemic schemes have found diverse applications in distributed computing: load balancing, topology management, audio and video streaming, computing aggregate functions, failure detection, network and resource monitoring, to name a few. To the best of our knowledge, our algorithm is the first attempt at exploring the potential of this paradigm to detect replicas in a wireless sensor network. Through analysis and simulation, we show that our scheme achieves robust replica detection with substantially lower communication, computational and storage requirements than prior schemes in the literature. Our goal is to test the potential of using epidemic algorithms in sensor network duplicate node detection, and we show that this technique is very naturally suited not only for sensor networks, but several other classes of networks, as well.

At this point, we would like to note that our work is focused only on detecting duplicates in the network. To make this clear, in this paragraph, we mention a couple of problems that are (in spirit) related to node replica detection. The Sybil attack in sensor networks refers to the ability of a sensor node to claim multiple identities. Some techniques to counter Sybil attacks in a sensor network include network testing, key space verification, node registration and related verification protocols [5,6]. The second related problem is that of detecting compromised sensor nodes in the network [7]. We do not discuss either of these related problems in this paper.

The rest of this paper is structured as follows: In Section 2, we survey related prior work. Section 3 presents the network and the adversarial model, followed by a discussion of epidemic diffusion in Section 4. We present our algorithm **Discard** in Section 5 and the security analysis and results in Section 6. Finally, we discuss potential open problems and future work in Section 8 and conclude in Section 9. A preliminary version of this work was presented in [8].

2. Prior Work

In this section, we survey some of the prior work in duplicate node detection in stationary sensor networks. While we outline some of the most important approaches and contributions in this area, this survey is not to be treated as an exhaustive overview of prior and related work.

The first schemes to appear in the literature on detecting replica nodes in a wireless sensor network were centralized schemes [2]. In these schemes, all of the nodes in the network were required to send their neighbors' identities and claimed locations to a central base station. The base station would then check for replicas using these neighborhood lists and detect duplicates followed by broadcasting relevant revocation messages to the network. Clearly, such a solution, despite its simplicity and accuracy, is extremely expensive in terms of communication. Furthermore, it creates a central point of failure and leads to energy depletion of the nodes immediately surrounding the base station. Therefore, distributed solutions are strongly preferred to address this problem. Other variants of such centralized solutions are discussed in [9–11].

Another scheme that is almost a folklore distributed detection scheme outlined in the research literature is a naive distributed detection scheme called node-to-network broadcasting (N2NB) [10]. Each node floods the network with an authenticated broadcast message containing its claimed location and stores the location claims of its neighbors. Upon receiving a conflicting claim (a node identity residing in multiple locations), this whistle-blower node signals the detection of a clone and alerts the rest of the network. Clearly, this scheme has a 100% detection rate, but incurs a high communication and storage cost. Wireless sensor motes have stringent memory and energy constraints, and thus, this solution is not satisfactory. Subsequently, other similar broadcast schemes were developed that trade detection accuracy for communication and storage costs [12].

Localized voting approaches were then proposed, which remedied some of these energy and memory issues. Using a voting mechanism, neighbors are required to reach a consensus on the legitimacy of a given node. However, these neighborhood voting protocols [1,5] failed to detect replicas distributed in disjoint neighborhoods of the network. By deploying replicas at least two hops away from each other, an adversary can evade detection from such schemes.

The first distributed and randomized replica detection schemes were proposed by Parno *et al.* [10]. Motivated by emergent algorithms [13], they proposed two related schemes. In these schemes, nodes broadcast their neighbors' identities and locations to other nodes in the network. They rely on special witness nodes in the network for duplicate detection. When a particular sensor node receives conflicting location claims and realizes that a specific node identity exists in more than one location, such a node becomes a witness for this duplicate. Once detected, the witness nodes broadcast this information to the rest of the network, and an appropriate revocation mechanism is triggered. The first scheme, called randomized-multicast, distributes location claims to a randomly-selected set of nodes in the network. With an appropriate choice of parameters, they show that the birthday paradox ensures that a collision occurs with high probability, thereby creating witness nodes. The second protocol, called line-selected multicast, takes advantage of the routing topology of the network to select witness nodes. They show that line-selected multicast is more efficient that randomized-multicast in terms of communication and storage requirements.

Inspired by the emergent algorithmic approaches, Conti *et al.* [14,15] proposed a randomized, efficient and distributed algorithm called REDthat improves upon the communication and storage costs and the detection accuracy of the line-selected multicast protocol proposed by Parno *et al.* [10]. RED executes at fixed intervals of time, and in each time period, witness nodes are selected as a function of a random seed broadcast by the base station. Because of the randomized nature of choosing witness nodes, the adversary is unable to identify and compromise these witness nodes ahead of time. While the costs of RED are lower than that of line-selected multicast per time period, the overall costs are higher, since RED repeats itself periodically.

Zhu *et al.* [16] proposed two variants of a localized-multicast scheme called single deterministic cell (SDC) and parallel multiple probabilistic cells (P-MPC). In these schemes, the witness nodes for a sensor are randomly selected from the nodes that are located within a particular geographic region in the sensor field. First, the scheme deterministically maps a node ID to a specific region in the field and then randomly chooses witnesses within this region. SDC and P-MPC differ in whether the location claims are broadcast to a single region or to multiple regions probabilistically. They show that these schemes achieve a higher detection rate than line-selected multicast with comparable communication costs.

Choi *et al.* [17] proposed a duplicate detection scheme called SETwith lower communication and storage costs than the line-selected multicast protocol proposed by Parno *et al.* [10]. SET detects clones by first treating the sensor network as non-overlapping regions and forming exclusive subsets in each of these regions. Then, the scheme performs set operations, namely unions and intersections, of these exclusive subsets in the network. If the intersections of these subsets are non-empty, this indicates the presence of a replica in those regions.

Based on the assumption that nodes in a sensor network are deployed in groups, Ho *et al.* [18] proposed a distributed, efficient scheme to detect replicas with lower communication, computation and storage overheads than the schemes listed earlier. They argue that this assumption is reasonable since many sensor networks are deployed in groups, either by dropping them over the field via airplanes or scattered by hand. Their scheme reduces communication overhead, since most nodes are able to communicate without generating location claims, as long as they are able to directly send messages to one of their group members. In our work, we make use of this assumption to design our detection scheme called Discard.

For a more comprehensive treatment of node replica detection schemes in wireless sensor networks, we refer the reader to the survey by Zhu *et al.* [12].

3. Model and Assumptions

In this section, we outline the network and the adversarial model that we employ in our work. Note that we use the graph theoretic terminology, node, throughout this paper to refer to a sensor mote. We adopt the network model proposed by Ho *et al.* [18] where the nodes in the network are deployed in groups and the adversarial classification model proposed by Conti *et al.* [14,15].

3.1. Network Model

We study the problem of detecting duplicates in a two-dimensional sensor network that is stationary, where the communication between nodes is bidirectional. In our framework, the sensor nodes are not mobile and, hence, do not change locations after being deployed in a two-dimensional sensor field. As is typical of most sensor networks, communication links between nodes are treated as bidirectional. The important assumption in our model, as adopted from Ho *et al.* [18], is that the nodes are deployed in groups. We describe this group deployment structure in the section below.

3.1.1. Group Deployment of Sensor Nodes

In this paper, we consider a sensor network deployment model where nodes are deployed in groups. The underlying expectation is that the nodes in the same group are geographically close to each other after deployment. This group deployment model is not new and has been used for various applications in sensor networks. Some examples include group-based key pre-distribution [19], public key authentication using group knowledge [20], localized anomaly detection [21], key management schemes in group deployed wireless sensor networks [22] and detecting duplicates in wireless sensor networks using group deployment knowledge [18].

The typical deployment procedure that is followed in such group-deployed applications begins with the sensor network architect partitioning the sensor field into territories by choosing appropriate deployment points in the field. Having chosen the deployment points, which amounts to having decided on the number of territories to partition the sensor field into, the architect gathers the sensor nodes into groups and assigns them to their intended territories. This assignment is done by programming the nodes in each group G with a group ID G_{id}, a unique id within the group U_{id} and their pre-assigned intended group deployment two-dimensional (x_G, y_G) coordinates. Keying materials are also pre-loaded onto each node using any of the techniques outlined in the literature [1,23–25] for pairwise key establishment. As is common in sensor networks, every message exchanged between sensor nodes is encrypted with this pairwise key and includes the ID of the sender in plaintext to thwart simple spoofing attacks. The nodes are then deployed, as groups of nodes, via an airplane to their pre-assigned group deployment points as the plane flies over these locations. While not all sensor networks are deployed exactly as described above, deployment models such as this are quite similar in spirit. In our work, we exploit this spatial relationship among sensors in a group to detect duplicates in the network.

Deployment Distribution

During the actual deployment, sensors naturally land at random locations in their territory. We use a two-dimensional Gaussian distribution used extensively in the literature [18,21,26,27] to model the likelihood of a node being a certain distance from its intended group deployment point. While we use the Gaussian distribution model in this paper for exposition purposes, our detection algorithm **Discard** does not preclude use of other probability distribution functions. Let (x_G, y_G) be the intended group deployment point for nodes in a group G. A sensor node $n \in G$, during deployment, lands at the coordinates (x, y) as given by the Gaussian distribution:

$$\begin{aligned} f(x,y) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-x_G)^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-y_G)^2}{2\sigma^2}} \\ &= \frac{1}{\sigma^2 2\pi} e^{-\frac{(x-x_G)^2+(y-y_G)^2}{2\sigma^2}} \end{aligned} \tag{1}$$

Here, (x_G, y_G) is the group deployment point for group G and σ is the standard deviation of the Gaussian distribution. We note that Equation (1) above denotes the probability distribution of having a node at a given point (x, y).

An Example

To illustrate this deployment paradigm, consider a square sensor field of side length 1000 units as used in the simulation experiments we conduct and by Ho *et al.* [18]. Suppose the sensor network architect decides to partition this field into territories that are "grids" with four rows and five columns, as shown in Figure 1. In this particular example, there are 1000 nodes; the communication range for each node and the standard deviation, σ, were both set to 50 distance units. These 1000 sensor nodes are now evenly distributed into these 20 groups with 50 nodes in each group. The group deployment points for each group are the center of the grid territory that the nodes were being deployed in, as shown in the figure. We use this particular network topology in our simulation experiments and for further discussions in this paper for ease of exposition, but we stress that the functionality of **Discard** is not limited to this particular grid topology.

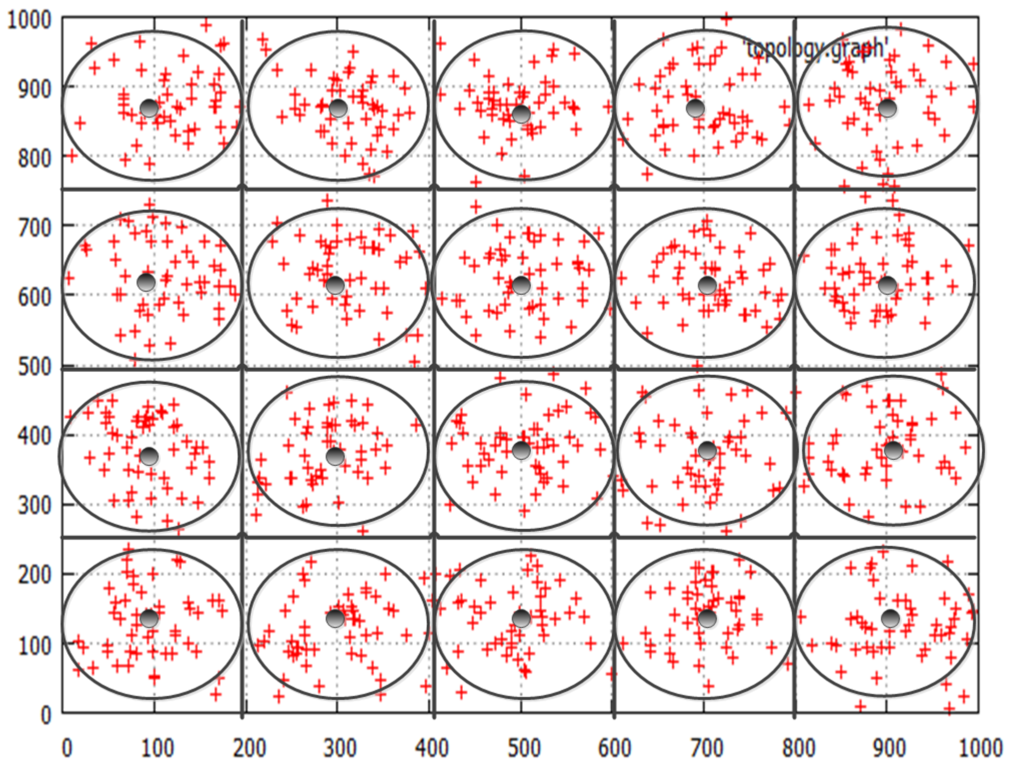

Figure 1. Group-deployed sensor network topology.

Home Zone

In our protocol, we distinguish between nodes that are deployed close to their intended deployment point from nodes that are located further away from their deployment point. To this end, we define the home zone of a group as the region of its territory that is no more than distance r units from its deployment point. In other words, the home zone of a group is defined as a circle of radius r, which is a parameter of the system, centered at the deployment point as shown in Figure 1. Nodes of a group G that are deployed within distance r, *i.e.*, within the home zone, are called safe nodes, and nodes that are located outside the home zone are called unsafe nodes. The terms safe and unsafe are meant to reflect the roles that these nodes play in our algorithm, as we describe in Section 5.

Intuitively, safe nodes are tasked with the responsibility of detecting and disseminating the identities of the duplicate nodes and are implicitly trusted as being genuine (not duplicates) until proven otherwise, while unsafe nodes are burdened with the task of having to prove themselves as genuine and are treated as duplicates until they have proven otherwise.

Population Density

Since we chose to model our deployment process as a two-dimensional Gaussian distribution, the probability of a node to reside in its home zone, and consequently be a safe node, can be expressed as $1 - e^{-\frac{r^2}{2\sigma^2}}$, where r is the home zone radius and σ is the standard deviation of the Gaussian distribution. Suppose, we choose the home zone radius r to be 100, as shown in Figure 1. Further, set the communication range of each node and the standard deviation σ to each be 50 distance units.

The probability of a node to be safe is about 0.86. Therefore, for a group consisting of 50 nodes, we expect to see about six unsafe nodes and 44 safe nodes in a particular group territory. In our empirical experiments, we observed that the number of unsafe nodes is about five per group on average, which implies that 45 nodes reside in their home zone. Indeed, if the deployment accuracy were poor, *i.e.*, a higher standard deviation for the distribution, we would find a greater number of unsafe nodes. Since our algorithm differentiates between the safe and unsafe nodes, the costs associated with our algorithm are quite dependent on the deployment accuracy. In this paper, we conduct our simulation experiments and present the costs associated with **Discard** for $\sigma = 50$, $r = 100$, and the communication range of each node at full power equals 50, as done in prior work.

Communication

Communication links between two sensor nodes are established based on the distance between the nodes. Two sensor nodes m and n are able to communicate with each other if the distance between the nodes is no more than their communication radius, and as noted above, this communication is bidirectional. From our empirical analysis, we note that the sensor network formed using the Gaussian deployment model, with parameters as listed in the previous paragraph, yields a connected (undirected) graph in the sensor field. We also note that while there may exist a communication link between two nodes m and n, whether or not the nodes actually exchange messages with each other depends on each node's determination of the other node's authenticity status and will be discussed in detail in Section 5.

Assumptions

In this paragraph, we list all of the assumptions we require in our scheme. As noted earlier, the deployment points are pre-determined before the actual deployment and can be easily inferred from the group IDs by the sensor nodes in the network. Hence, nodes are incapable of claiming an incorrect deployment point, even if they are compromised (assuming of course that the adversary is not capable of generating new "Sybil" identities, which is a tacit assumption in prior work). In addition to keys that are pre-loaded to enable pairwise key establishment as described earlier, every sensor node is also equipped with keys to generate digital signatures. We use an identity-based public key scheme popularized by several replica detection schemes in the literature [10,14–16,18] and that has been shown to be feasible in the current generation of sensor motes [28–30]. Lastly, we assume that an attack-resistant location estimation protocol has been implemented in the network using one of the schemes outlined in the literature [31–33]. This permits a sensor node to discover its actual location coordinates after deployment. This assumption is quite reasonable and has been used by existing node replica detection schemes in the literature [10,18].

A final assumption we require is that nodes under the adversary's control (both compromised nodes and their replicas) follow the procedures expected by **Discard**. This has also been tacitly assumed by existing schemes outlined earlier.

3.2. Adversary Model

In this section, we describe the characteristics of the adversary who intends to inject replicas into the sensor network. We adopt the adversary classification model of Conti *et al.* [14,15] and identify the particular class of adversaries that our protocol **Discard** defends against. Conti *et al.* [14,15] classified adversaries based on their abilities along two attributes.

1. Mobility: The adversary was classified as being either localized or ubiquitous based on his mobility. A localized adversary was limited to injecting replicas within a particular convex sub-area of the sensor field while a ubiquitous adversary had the freedom to traverse the entire network field in his effort to identify, compromise and inject duplicate nodes. In some sense, this captures the degree of physical monitoring or surveillance present in the sensor field. If the sensor network is highly unattended, then perhaps this allows the adversary to freely traverse the entire field without fear of detection. It is of course in the adversary's best interest to operate in a stealthy manner and attempt to avoid detection, since detection could trigger automated protocols, such as SWATT [34], to sweep the network and remove compromised nodes.
2. Intelligence: The adversary was classified as being either oblivious or smart based on his or her intelligence to identify potential target nodes to compromise. The smart adversary chooses to compromise sensor nodes that yield the most benefit, *i.e.*, that maximizes the chances of his or her injected replicas to go undetected, while the oblivious adversary merely compromises a random sensor node oblivious of the processes used in the detection algorithm.

As we will show, our scheme defends against the most powerful adversary according to the above classification model: the smart/ubiquitous adversary. Given our network setup and the classification of nodes as being in the home zone (safe) and being outside the home zone (unsafe), we can succinctly describe all possible actions of such an adversary as described here. We use the term source to refer to the original sensor node chosen by the adversary for compromise and subsequent duplication. The possible adversarial choices for the source node are to either choose a node from inside the home zone (IHZ) or outside the home zone (OHZ) from any territory. After compromising the source node, duplicate nodes can now be deployed either in their corresponding home zones (IHZ) or elsewhere (OHZ). The third possibility to consider is what the adversary does with the original source node. The original sensor node can either be removed from the network (REM) or left intact in its initial deployment point (inserted (INS)). The notation "INS" stands for the duplicate nodes being "inserted" into the network with the original node not being removed. Finally, we are only interested in studying those scenarios when there are at least two nodes with the same identity in the network (in each of the eight cases). In other words, we are not trying to identify compromised nodes, but only to detect the duplicate nodes in the network. Therefore, we assume that the adversary, after his duplication efforts, has at least two copies of the same node identity in the sensor network at distinct locations. Indeed, if there are more replicas, **Discard** will only detect them faster. In our simulation experiments, we analyze (conservatively) the performance of **Discard** in the presence of exactly two duplicates to best understand the costs and other performance characteristics.

4. Epidemic Diffusion

Our goal in this paper is to explore the potential of emergent, epidemic algorithms on detecting duplicate nodes in a wireless sensor network and to design an appropriate detection algorithm using these algorithmic techniques for this problem setting. Epidemic algorithms and their applications to distributed computing were popularized by Demers *et al.* [35] as applied to the problem of maintaining distributed, replicated databases. These algorithms have been recognized by the distributed computing community as a robust, reliable and scalable means to disseminate information among the distributed computing nodes, similar to the manner in which an epidemic spreads through a distributed, connected population of people. They have also been applied in several other areas of distributed computing, such as load balancing, topology management protocols, audio and video streaming, computation of a large set of aggregate functions, network and resource monitoring, failure detection, to name a few. A survey of epidemic algorithms and their applications to distributed computing can be found in the work presented in [36,37]. Of interest to note is the fact that our problem setting may be viewed as the dual problem of efficient mule collection of network coded data that has been studied extensively in the literature [38]. In the efficient data mule collection problem over sensor networks, one tries to improve data resilience to node failures and data collection by mobile sensor mules by employing erasure coding, such that a small subset of the sensor nodes stores some linear combination of all of the network data. In contrast, we attempt to disseminate sensed network data among many sensor nodes using epidemic gossip so as to enhance duplicate detection.

In this paper, to the best of our knowledge, we initiate the study of using these techniques to detect duplicate nodes in a wireless sensor network and show that epidemic algorithmic techniques offer a very natural approach to solve this problem. To this end, we design our duplicate detection scheme, which we call **Discard** , as a variant of a couple of interesting epidemic algorithms discussed below. Furthermore, we designed our scheme to ensure that it has an effective and robust replica detection capability with low communication, computation and storage overheads, as this is imperative in a wireless sensor network to conserve node energy and make efficient use of limited node storage and processing power.

In our problem setting of detecting replicas in a wireless sensor network, our aim is to design schemes that enable sensor nodes to identify the duplicate nodes in the network by gossiping and exchanging information with each other. To this end, we have identified two primary tasks that need to be accomplished through the design of appropriate epidemic mechanisms. We list these two tasks below and discuss them briefly below.

1. Primary epidemic: Sensor nodes should detect a duplicate node in their midst by exchanging neighborhood information with each other.
2. Secondary epidemic: Once a duplicate node has been detected by a sensor node, this information should be disseminated to the rest of the nodes in the network as soon as possible.

We call the above two epidemics primary and secondary merely to reflect the fact that the secondary epidemic (identity of the duplicate nodes) can be spread only after the primary epidemic algorithm has been successful in detecting a duplicate node by exchanging the neighborhood information.

In this paragraph, we discuss, very briefly, the goals of each of these epidemic mechanisms and relegate the details to Section 5. Each sensor node maintains a neighborhood cache of a particular size. This cache reflects the nodes' knowledge of its neighbors. This cache not only maintains information about the nodes' immediate neighbors, but also of the nodes beyond its communication radius. Specifically, this cache contains the identity (group ID and unique ID of the node within the group) and location information of the nodes in the network. Periodically, the sensor nodes exchange and merge their neighborhood caches amongst each other (dissemination of the primary epidemic) in an effort to detect the presence of a node identity in two different locations. Such a node is clearly a duplicate. Having identified a duplicate node, the identity and location of this duplicate node is added to its duplicate cache. The sensor nodes then exchange and merge the duplicate caches with each other, hence serving to propagate the secondary epidemic. We would like the identities of the duplicate nodes to be diffused through the sensor network to all of the other nodes in an efficient, reliable, robust and scalable fashion. The contents and structure of these caches, how they are populated and the exchange mechanism are some of the questions that need to be answered, and we discuss this in Section 5.

Having identified the two fundamental epidemic mechanisms that we would like to design, we now need to decide which epidemic algorithms to use in designing our scheme. There are a couple of choices available to us as described in the literature [35,36]. We adopt the terminology used in the epidemic literature and refer to any update or information that a node wishes to share with the rest of the network as an infection. A node that has not received the infection is called susceptible, and a node that is infected, but has stopped spreading it, is called removed. Of course, contrary to the goal of the epidemiology studies, we would like to ensure that the infection of interest indeed spreads rapidly and preferably completely through the network like an epidemic. In addition, unlike the epidemiology studies, we are not restricted to modeling a particular epidemic. Rather, we are free to design an epidemic spreading mechanism that is most efficient in spreading the infection of interest under the constraints imposed by the sensor network. In our case, the infection(s) that we would like to spread are the neighborhood and duplicate caches of each sensor node at periodic time intervals. Our tasks therefore are to study the fitness of the following epidemic mechanisms for our problem setting followed by designing an appropriately customized scheme to detect duplicates in the sensor network. We now discuss the sub-class of epidemic algorithms known as epidemic-diffusion. Epidemic-diffusion aims to achieve the algorithmic equivalent of "hearing something through the grapevine" in our everyday social settings. In particular, we discuss, very briefly, the following three diffusion algorithms presented by Demers *et al.* [35] when applied to the context of wireless sensor networks.

- Direct mail: Each sensor node, upon receiving an infection, immediately transmits this infection to all the other nodes in the network. This method is of course very timely. However, it is quite expensive in terms of communication and node storage. Even assuming that every sensor node possesses complete knowledge of every other node in the network, there are issues of message loss and the bottleneck proportional to the size of the network created at each sending node. Instead of direct mail, one could perhaps use a broadcast mailing mechanism to alleviate these issues. However, such broadcast mailing schemes would necessarily have to rely upon and take advantage of the knowledge of the distributed sensor network and would probably be designed using one of the epidemic primitives discussed below. In fact, such broadcast schemes have been used extensively in earlier duplicate detection in sensor network research, as discussed in Section 2. This is, in spirit, essentially a folklore broadcast distribution mechanism implicit in previous work.

However, these prior research works do not explicitly describe the broadcast mechanism used, nor do they precisely analyze the cost incurred by their broadcast schemes, in terms of computation, communication and storage. The costs associated with such broadcasting are typically dismissed by prior researchers as negligible, so long as these broadcast messages are contained within a reasonably small geographic region. In this paper, we account for every interaction between nodes and measure the communication, computation and storage costs precisely.

- Anti-entropy: Each sensor node n periodically contacts another node p at random, and they exchange infections with each other. This exchange can be accomplished in a couple of ways: push, infections known to n, but not to p are sent from n to p; pull, infections known to p, but not to n are sent from p to n; or push-pull, bidirectional information/infection transfer. It has been shown that either pull or push-pull is greatly preferable to push [35] in terms of dissemination speed, and hence, we opted to use the push-pull mechanism in **Discard** for diffusing the primary and secondary epidemics and describe it in more detail below.
- Rumor mongering: Each sensor node is initially susceptible (has no infection to share). When a node gets infected, this infection, called a rumor, is periodically transmitted (pushed) to a random sensor node in the network. When the node has tried to push its rumor a certain number of times, the node stops transmitting its rumor and is effectively removed in the sense that it becomes inactive. There are a couple of variants proposed by Demers *et al.* [35], which determine when a node is removed:
    1. Feedback *vs.* blind: A sender node is removed with some probability only if it has pushed its rumor to a recipient that already knows the rumor. In blind mode, the sender is removed with some probability regardless of whether the recipient is aware of the rumor and is perhaps better suited in a sensor network scenario, as we could reap some energy/communication savings by eliminating the feedback message from the recipient. Nevertheless, we do not use this scheme and explain our reasoning below.
    2. Coin *vs.* counter: In the coin mode, the sender node is removed with some probability based on the flip of its coin. In the counter mode, the sender node is removed with probability one after a certain number of rumor pushing attempts.
    3. Typically, the feedback/blind and coin/counter modes are combined to obtain schemes, such as coin-blind and feedback-counter modes of operation, for the rumor mongering diffusion scheme.

As noted above, we chose to implement the push-pull, anti-entropy epidemic-diffusion mechanism to diffuse both the primary and secondary epidemics in the sensor network. Our justification for our choice is the following: Recall the operational states of typical sensor nodes, such as the Mica2 motes: transmit, receive, idle and sleep. In their work on energy analysis of nodes in a sensor network, Xu *et al.* [39] showed that most sensor node transceivers operating in the idle mode have a power consumption almost equal to the power consumed in receive mode. Hence, it is energy efficient to completely shut down the transceiver rather than leave it in the idle mode when not transmitting or receiving. They also showed that a significant amount of power is consumed when switching from sleep mode to transmit mode in order to transmit a packet. During transmit and receive, the microcontroller is powered on along with the wireless transceiver. As an aside, it is interesting to note that the energy cost of computation is small when compared to data transmission [40]. Hence, from these above observations, it is clear that rumor mongering would consume greater power due to its repeated transmit/idle/sleep cycles for every update, and so, we did not choose this option. We therefore chose anti-entropy for propagating the primary epidemic, *i.e.*, the neighborhood caches that need to be exchanged among the nodes to detect a duplicate node. Having used anti-entropy to transmit the cache to the recipient, we decided to use anti-entropy for exchanging the duplicate node cache among these two nodes, as well. We note that our discussion on epidemic algorithms is not meant to be exhaustive. Our objective is to show that epidemic-based schemes are efficient at detecting duplicate nodes and to bring attention to these schemes to the sensor network community. Our hope is that this work will inspire future work and exploratory studies on other epidemic algorithms and their fitness for solving this and similar problems. We note that there are several algorithms, mainly in the peer-to-peer (p2p) literature, such as lpbcast [41] and Newscast [42], used to build self-organizing topology management protocols in large and dynamic p2p environments, which are built using rumor mongering epidemic techniques.

5. Our Epidemic Algorithm

Before we describe Discard in detail, we first present the intuition behind the algorithm. Recall from Section 3.1.1 that the nodes in the network are equipped with a location estimation protocol, which enables them to determine where they are located and to estimate their distance to their group deployment point. Furthermore, recall that nodes deployed within distance r from their group deployment points are called safe and those deployed further away as unsafe. Hence, each node is able to determine if it is safe or unsafe, and every sensor node chooses to communicate with only those nodes that it trusts. Therefore, a node accepts messages from the trusted home zone members of its own group and authenticated nodes located outside their home zone.

5.1. Overview

We explain the concept of trust in more detail in this section. Let us first consider the case when node n is located within its home zone. Suppose that the in-home-zone sensor node n (a member of the group G) receives a message forwarding request from a neighboring node p. If node p is also a member of the group G, is safe (within the home zone, *i.e.*, the distance between node p and its deployment point is less than r) and node p is not an element of node n's duplicate cache, then node n believes that p is a trusted neighbor and agrees to the message forward request. On the other hand, if node p (regardless of whether it is in group G) is outside n's home zone, node p sends node n evidence of its authenticity along with the message forward request. The message forward request succeeds only upon verifying node p's authenticity based on the presented evidence. If the verification results in node n learning that p is spurious, it ignores all messages from p. We will discuss the details of the structure and format of the evidence presented by node p in Section 5.2 below.

Let us now analyze the case when node n is located outside its home zone and it receives a message forward request from node p. The node p is either within its home zone or outside. In either case, node p will be required to send node n evidence of its authenticity. As before, if the check succeeds, the message forward request will be fulfilled. Otherwise, all messages from p will be ignored by n. Of course, under no circumstances will node p initiate any communication with n if it believes that n is a replica.

Intuitively, a node that is not near its own group members will be isolated and unable to send messages. The goal is to ensure that we do not introduce any significant communication, computation or storage overhead. Each node immediately determines whether to forward messages from other sensor nodes by only looking at their IDs and actual deployed locations. An important point, arising from communication cost considerations, that we would like to stress is that these actions are made by node n and p just once, whenever the node p first contacts n.

5.2. Discard Duplicate Detection Scheme

In this section, we describe the functionality of the **Discard** algorithm in detail. The pseudocode presented in Algorithms 1 and 2 is modeled by two distinct threads executed at each sensor node. For the sake of exposition, we describe the algorithm as an interaction between two sensor nodes n and p. In particular, we will discuss the interaction between the active thread of node n and the passive thread of node p. The primary distinction between these two threads is captured by the fact that while the active thread actively takes the initiative to communicate, the passive thread merely responds to incoming messages. The active thread repeats itself periodically, denoted by δ time units in the algorithm.

Each sensor node maintains two caches, the neighborhood cache and the duplicate cache. For instance, the neighborhood cache of node n contains information about n's neighbors and their location, *i.e.*, their (x, y) coordinates. Node n's duplicate cache contains information (ID and no more than two distinct locations) about the duplicate nodes in n's home zone that n is aware of at a particular time instant. The structure of these caches and the interaction between two sensor nodes n and p is depicted in Figure 2. The neighborhood cache is of size c, and the duplicate cache is of size d. The effect of the interaction between these two threads is essentially a continuous random shuffling of the caches of the sensor nodes within the home zones of the sensor network, with requests and replies exchanged between the nodes. Here, we discuss the algorithm in terms of the parameters $\delta, c,$ and d and relegate the discussion of the choice of these parameters to Section 6.

Let us now discuss the interaction between the two threads: the active thread of node n, $n.active()$ and the passive thread of node p, $p.passive()$, the two nodes that are involved in the cache exchange and duplicate detection process. In this paragraph, we discuss the procedure where node n is located within its home zone. We use $NCache$ and $DCache$ to refer to the neighborhood and duplicate cache of a sensor node. Furthermore, we use the dotted notation, borrowed from object-oriented programming, to refer to membership. Hence, $n.Ncache$ and $p.Dcache$ refer to the neighborhood cache of node n and the duplicate cache of node p, respectively. At every δ time interval, the active thread of node n seeks out an immediate, genuine (non-duplicate) neighbor p in its home zone with which to interact. By an immediate neighbor, we mean a sensor node p that is within the communication radius of node n and is also within n's home zone. Furthermore, it is important to note that n will ensure that the chosen node p is genuine (not a duplicate), *i.e.*, p is not contained in n's duplicate cache. Our reason for enforcing nodes to interact only with immediate, in-home-zone nodes stems not only from the fact that we need to conserve energy during the cache exchange process, but also because each home zone is tasked with the responsibility of detecting duplicate nodes from their territory in the entire network. This division of labor, as we will show, has a profound effect on the communication cost. Having chosen a peer node p, n sends its neighborhood cache $< NC, n, n.NCache >$ to p (Line 8 of $n.active()$ in Algorithm 1). The passive thread of node p responds to this message with its own neighborhood cache $< NC_{rep}, p, p.NCache >$ after ensuring that n is not a duplicate, as indicated by its duplicate cache records, (Line 4 of $p.passive()$ in Algorithm 2). The message format $< NC, n, n.NCache >$ follows the structure $<$ type of message, sender, content $>$, and the subscript rep stands for reply.

Algorithm 1: Discard duplicate detection protocol: active thread.

```
Data: Protocol run by sensor nodes $n$ and $p$.
Result: Caches updated and duplicates detected.
1  Active Thread n.active()
2      while true do
3          wait $\delta$ time units
4          if $n$ is safe, i.e., within its home zone then
               // pick a peer neighbor $p$
5              repeat
6                  $p = n.getRandomHomeNber()$;
7              until $p \notin n.DCache$;
               // send caches to peer
8              send $< NC, n, n.NCache >$ to $p$;
9              on receive $< NC_{rep}, p, p.NCache >$
10                 $n.detectAndUpdateDCache()$;
11                 $merge(n.NCache, p.NCache)$;
12             send $< DC, n, n.DCache >$ to $p$;
13             on receive $< DC_{rep}, p, p.DCache >$
14                 $merge(n.DCache, p.DCache)$;
15         end
16         if node $n$ is unsafe then
17             Random seed $rs$ ←baseStation;
18             for $i \leftarrow 1$ to $v$ do
19                 $verifier_i \overset{rs}{\longleftarrow}$ {$n$'s home zone};
                   // $n$ contacts $v$ verifiers
20                 send $< verify, n >$ to $verifier_i$;
21             end
22         end
23     end
```

Algorithm 2: Discard duplicate detection protocol: passive thread.

```
Data: Protocol run by sensor nodes $n$ and $p$.
Result: Caches updated, duplicates detected.
1  Passive Thread p.passive()
       // reply to sender-neighbor cache
2      on receive $< NC, n, n.NCache >$
3      if $n \notin p.DCache$ then
4          reply $< NC_{rep}, p, p.NCache >$ to $n$;
5          $p.detectAndUpdateDCache()$;
6          $merge(n.NCache, p.NCache)$;
7      end
       // reply to sender-duplicate cache
8      on receive $< DC, n, n.DCache >$
9      if $n \notin p.DCache$ then
10         reply $< DC_{rep}, p, p.DCache >$ to $n$;
11         $merge(n.DCache, p.DCache)$;
12     end
       // reply to sender-verification
13     on receive $< verify, n >$
14     if $p.isVerifier()$ then
15         if $n \notin p.DCache$ then
16             send $Sig_p(genuine)$ to $n$;
17         end
18         else
19             send $Sig_p(duplicate)$ to $n$;
20         end
21     end
```

Now that both nodes have exchanged each others neighborhood caches (primary epidemic), they each run the routine $detectAndUpdateDCache()$. This subroutine, as its name implies, traverses both the caches and detects if there are nodes in their home zone with the same ID, but different locations. If found, this node is clearly a duplicate, and this node's ID and location information is inserted into their duplicate cache stores. Both n and p now merge each others neighborhood caches using the subroutine $merge(\cdot, \cdot)$ as indicated in Line 11 and Line 6 of the active and passive threads in Algorithms 1 and 2, respectively. This is merely an independent, shuffling process as described here: for a neighborhood cache of size c, first, $c - 1$ randomly chosen entries from the collection of both caches (of size $2c$) are chosen to populate each participating node's $NCache$ repository. Next, each node also inserts its own ID and location into this cache. This step is important to ensure that the cache gets populated with each node's own entry prior to cache exchanges and also essential to bootstrap this exchange process. Next, a similar cache exchange and merge procedure for the duplicate caches is performed by n and p, as indicated in Lines 12–14 and 8–12 of the active and passive threads respectively (secondary epidemic).

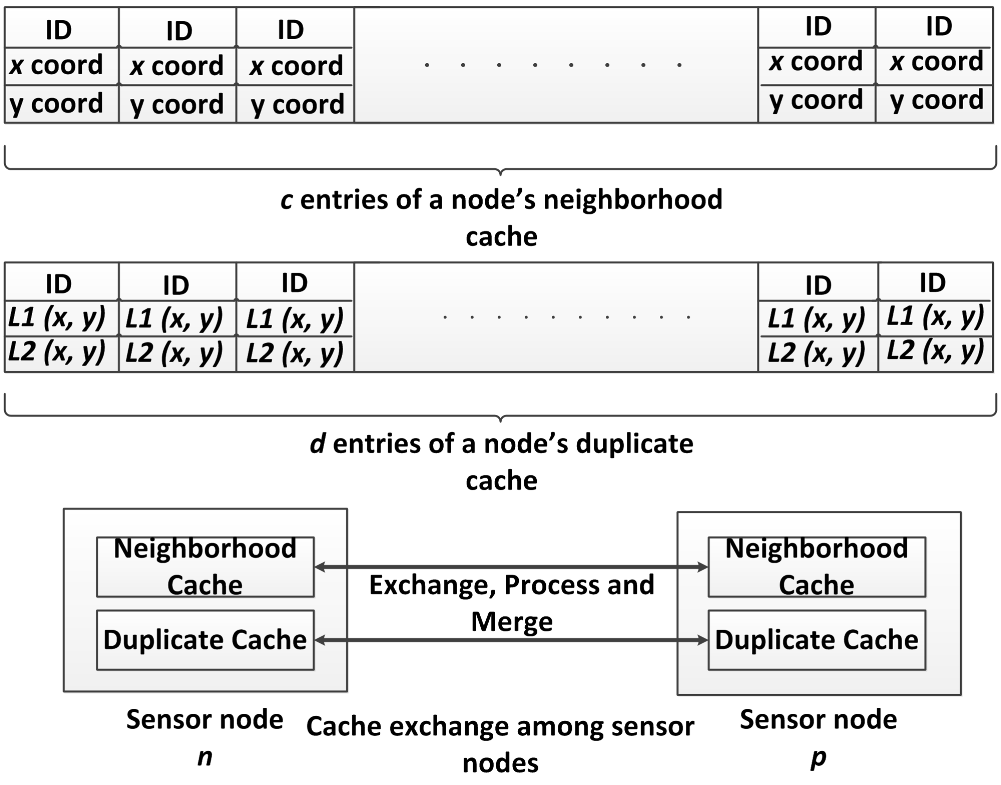
ID
x coord
y coord
c entries of a node's neighborhood cache
ID
L1 (x, y)
L2 (x, y)
d entries of a node's duplicate cache
Neighborhood Cache
Duplicate Cache
Exchange, Process and Merge
Sensor node n
Cache exchange among sensor nodes
Sensor node p

Figure 2. Cache structure and exchange. Note that the duplicate cache only needs to store the identity of the duplicate node and no more than two distinct locations regardless of the number of locations in which the replica has been deployed.

We now discuss the procedure when node n is located outside its home zone. Since the above cache exchange process was limited to interactions among genuine nodes within the home zone, nodes outside their home zone will need to provide evidence of their authenticity in order to participate in network activities. As noted earlier, node n is aware that it is located outside its home zone and follows the procedure discussed in this paragraph, listed in Lines 16–22 of the active thread in Algorithm 1. In each time period, the base station broadcasts a random seed that is unique for every time period. Once node n receives this seed, it determines which nodes in its home zone act as verifiers at this time instant and contacts them. Having determined the identities of the verifier nodes, node n sends its signed $< verify, n >$ message to these verifier nodes and awaits their response. The $verify$ message contains the ID and the (x, y) location of the sender node. Having received this message, the verifier checks to see if this sender at this location is a duplicate. This is accomplished by simply checking if this sender is an element of its $DCache$, the duplicate cache. The verifier nodes also check to ensure that they are indeed chosen to be the verifier for this time period. In our algorithm, we require that each node outside its home zone approaches v verifiers, which is a parameter to be determined later to balance security and communication cost.

Upon receiving the $< verify, n >$ message (Lines 13–21 of the passive thread), p checks if it is a verifier, then responds to n with the appropriate signed response. If node n is not an element of the verifier's duplicate cache, a signed $genuine$ response is sent; otherwise, a signed $duplicate$ response is sent. The response $genuine/duplicate$ is merely an authenticity verification message containing the identity of the node n and its (x, y) location signed by p's private key. Observe that such a verification needs to be done by an outside-home-zone node n only once, and once successfully authenticated, it can begin participating in network activities with its neighbors. Furthermore, observe that there is no benefit that accrues to the adversary by cloning a node n with its $genuine$ signature and deploying it elsewhere. This is because the signature is tied to the node identity and the (x, y) location. When cloned and deployed at another location, this signature would no longer be valid for that $(ID, location)$ instance pair and, hence, would fail verification when presented as evidence to a potential neighbor. Furthermore, note that the node needs to present at least v genuine signatures to a potential neighbor to be accepted to participate in network activities. In case a node is unable to gather the required v signatures in one time period, it collects the signature responses it has received for this period, is forced to seek authenticated signatures in the next time period and is effectively cut-off from the network until that time. We discuss the reason for choosing v verifiers and the security implications in Section 6 below.

Finally, there is one additional scenario that we need to discuss. This is the case when node n that belongs to its home zone intends to communicate with a node p from a different territory. This is a typical occurrence of nodes deployed at the fringes of two territories. Before initiating such communication, node p will perform an authenticity check on the evidence that node n will present prior to communicating, as discussed before. As noted earlier, this verification check is performed only once.

In addition to the neighborhood and duplicate caches, each node also maintains a temporary cache called the $outsideHomeZone$ cache. The size of this cache can be small, as this cache is a temporary working cache and not used for persistent storage across time. This cache is, in spirit, a working storage area, to store the IDs and locations of nodes deployed outside the home zone. When a node n deployed outside its home zone sends a $< verify, n >$ message, the verifier nodes store the ID and location information for node n in this temporary cache. This helps detect duplicate nodes in the event that the adversary decides to duplicate a node from the home zone, deploy it elsewhere and remove the original source node from the home zone.

6. Security Analysis and Results

In this section, we present the security analysis and the costs associated with **Discard**. Recall that there are two epidemic processes at play in the sensor network employing **Discard**. The primary epidemic serves to disseminate the neighborhood information among the nodes, while the secondary epidemic spreads the identity and location information of the duplicate nodes discovered from the spread of the primary epidemic. The spread of the primary epidemic enables the sensor nodes to discover the presence of the same node identity in their midst at different locations, thereby locating the duplicate nodes in the network, which is subsequently disseminated in the secondary epidemic. It is a basic result of epidemic theory that starting with one infected node, the infection spreads rapidly across the entire population of nodes. In particular, this is achieved in expected time proportional to the logarithm of the population size [35]. Here, we note that the result in [35] about the spreading of infections in $\log(n)$ steps, where n is the number of sensors in the network, is valid under certain constraints, and a "geometric graph" of sensors communicating via wireless interfaces leads to a $\sqrt{n}$ propagation time. During the neighborhood cache exchange and merge process, some node eventually discovers the presence of a duplicate node in its midst. If the duplicate node were deployed outside the home zone, then one of the home zone nodes will learn of this replica when the duplicates contact this home zone node for acquiring its verification evidence. If the replica were deployed inside the home zone, the discovery process happens during the cache exchange and merge process. Let us now discuss the speed with which this duplicate node is detected and this information gets propagated through the network starting from one infected witness sensor node.

6.1. Detection Time and Accuracy

We first note that the underlying assumption in the "secondary infection" analysis that we present below is that the method in Line 6 of the active thread in Algorithm 1, *i.e.*, $n.getRandomHomeNber()$, returns a non-duplicate node p chosen uniformly at random from among all nodes in n's home zone. This assumption does not hold in our implementation, since we insist on cache exchanges occurring between nodes that are within communication range to conserve energy and the fact that a sensor node is not aware of all of the other nodes in its home zone. Therefore, in our implementation, p is not a node chosen uniformly at random from the entire home zone, but a node chosen uniformly at random from among n's immediate neighbors, *i.e.*, its communication radius. We present our analysis below, feigning that the required assumption above is fulfilled, but present empirical validation subsequently to justify our analysis. The results are presented in Figure 3.

Secondary infection analysis: Here, we analyze the speed with which the secondary infection spreads throughout the home zone. Here, secondary infection is meant to denote the identity and location of a duplicate node in the duplicate cache. The analysis below is adapted from the work done by Demers *et al.* [35] on replicated databases. In this analysis, we assume that this infection initially exists at only one node in the home zone. Of course, if there were more infected nodes, this spread would be accomplished faster. Let p_i denote the probability that a node n that belongs to the home zone of group G does not know the identity of the duplicate node at time i. Let s be the size of node n's home zone. Since we assume that only one node in the home zone is infected at the beginning of time, $p_0 = 1 - 1/s$. We are interested in expressing the quantity p_i as a function of time to understand how fast it decreases with time. To this end, let us begin by expressing p_{i+1} in terms of p_i. Node n will not get the infection at time $i + 1$ if any of the following conditions hold:

1. If n was not infected at time i,
2. The peer sensor node p that n contacted at time i was not infected and
3. No peer nodes that were infected contacted n at time i.

Let us now compute each of these probabilities. The probability of n not being infected at time i is p_i. The probability that p was not infected at time i is also given by p_i. Finally, let us compute the probability that no peer nodes that were infected contacted n at time i. At time instant i, there are $(1-p_i)s$ expected number of nodes that are infected. The probability of one of these nodes not choosing to contact n is $1 - 1/s$, since we assume that n was chosen uniformly at random from among all of the s nodes in the home zone. Hence, the probability that no peer nodes that were infected contacted n at time i can be given by $(1 - 1/s)^{(1-p_i)s}$. Putting all of this together, we see that:

$$p_{i+1} = p_i p_i (1 - 1/s)^{(1-p_i)s}$$

It follows therefore that,

$$p_{i+1} < p_i^2 < p_0^{2^{i+1}} = (1 - 1/s)^{2^{i+1}}$$

hence showing that p_i decreases super-exponentially fast. In Figure 3, we see the convergence of p_i for different sizes of the neighborhood caches exchanged among the nodes in the home zone corresponding to group G along with the theoretical expected exponential decline. With increasing cache sizes exchanged among the nodes (naturally increasing the communication cost), the discovery of duplicate nodes happens much faster, which leads to a faster dissemination of these duplicate node identities among the home zone nodes, as well. As we can see from the figure, in reality, the convergence is faster than what was suggested by our theoretical analysis. This is because, in practice, it is possible that a node disseminates the duplicate node identity in the same cycle that it learned about it, for which the theoretical analysis fails to account. What we can glean from this analysis is that, for the smallest cache size $c = 5$, all of the nodes in the home zone of the duplicated node (regardless of where the duplicate is deployed) learn about the replication within seven time cycles. The largest gain is towards the end of the time period, as one would expect from this randomized process. Observe that only about 50% of the nodes in the home zone are made aware of the replica after five time cycles where Time 0 corresponds to the time instant that the adversary inserts one of his duplicates into the network (either inside or outside the home zone). This implies that the adversary may be able to outsmart **Discard** for no more than seven time cycles before being detected. This advantage can clearly be controlled by increasing the cache size. Of course, realizing the importance that the time period plays in the adversary's advantage, the network architects would need to decide on an appropriate time period δ, as well. We would like to note that while the prior work results are meticulous in presenting their detection accuracy, they do not, however, present the relationship with time, as we show in our analysis.

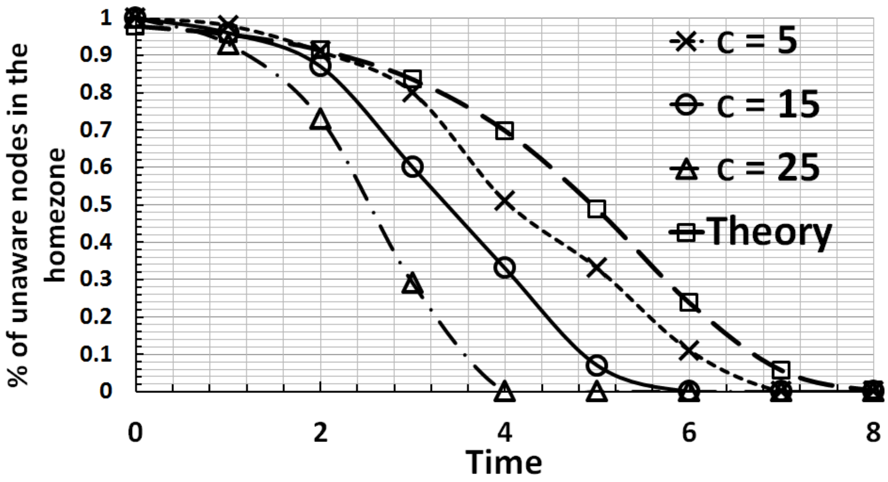

Figure 3. The proportion of nodes that have not learned about the duplicate, as a function of consecutive cycles of the **Discard** algorithm. The expected size of the home zone is 45 nodes. The plot points are averages of 100 runs.

6.2. Communication Cost

As we have seen, sensor node n initiates one outgoing active connection to a peer node p to perform its cache exchange (per cycle). We also need to determine the number of incoming connections for node

n in order to compute the total communication cost. Let b represent the average degree of a node, *i.e.*, the number of neighbors in the immediate communication range of the node. Let $N(n)$ refer to the set of neighbors of node n, and from the above assumption, $\mathsf{E}\left(|N(n)|\right) = b$. Each neighboring node of n may choose to conduct a cache exchange with n with probability $1/b$. As there are b such neighbors, the expected number of incoming contacts for node n is therefore one per cycle. This suggests that it might follow a Poisson distribution with mean one, and we verified this empirically, as displayed in Figure 4. We calculated the empirical probabilities in the following manner: we fixed a specific node from the 1000 deployed nodes in the sensor field. Having fixed this node, we tallied the number of incoming connections in each cycle for 1000 cycles and computed the probability to notice that they are very similar to that of the Poisson(1)distribution.

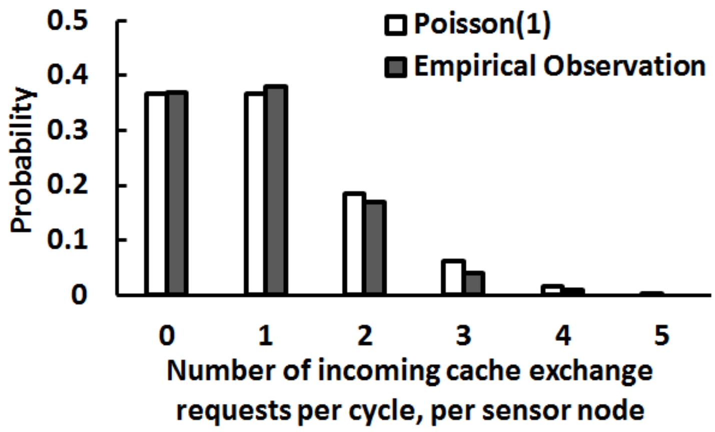

Figure 4. Number of incoming cache exchange requests per cycle, per sensor.

Therefore, in each time cycle, there are two cache exchanges: one outgoing and one (expected) incoming exchange. Recall that the neighborhood cache is of size c and the duplicate cache is of size d (refer to Figure 2 for the structure and content of these caches). The communication cost per node is therefore proportional to $2(c + d)$ every δ time units. More precisely, the cost per node, per δ time unit, is given by $T_{\$}(c + d) + R_{\$}(c + d)$, where $T_{\$}$ and $R_{\$}$ are the transmission and reception costs, respectively. In an effort to put these costs in perspective, let us now use the energy model proposed by Wander *et al.* [40] to compute the energy expended by $\mathbf{Discard}$ towards communication. In their model, the total available node battery is 324,000 mJ; 45.0 mJ for signing a packet (or signature check), 15.104 mJ for transmitting a packet, 7.168 mJ for receiving a packet (assuming a packet length of 32 bytes), 0.059 mJ for bit sending and 0.028 mJ for bit receiving. Assuming (using standard bit representation schemes in Mica2 motes) the ID field takes 16 bits and the (x, y) coordinates take 32 bits, c entries in the neighborhood cache constitute $48c$ bits and d entries of the duplicate cache constitute $80d$ bits. Concretely, for $c = 5$ and $d = 5$, we see that the neighborhood and the duplicate cache call for 240 and 400 bits exchanged each period. We can therefore estimate the cost using the model above to yield: $T_{\$}(c + d) + R_{\$}(c + d) = T_{\$}(640\text{ bits}) + R_{\$}(640\text{ bits}) = 37.76\text{ mJ} + 17.92\text{ mJ} = 55.68\text{ mJ}$ per node, per cycle. By adjusting the parameters c, d and δ, we can tradeoff between the time advantage that the adversary obtains and the lifetime of the sensor network in terms of the energy expended by the $\mathbf{Discard}$ algorithm for communication.

In our analysis above, we have not accounted for the one-time energy expenses incurred by the process involving the nodes deployed outside their home zone transmitting their signed locations and requesting verification evidence from nodes in the home zone. Suppose there are t nodes that find themselves deployed outside their home zone, and they contact v verifiers. We can estimate this cost as follows: Each of these t nodes transmit v signed location claims to the v verifiers. The v verifiers send their signed response back to these nodes. The costs incurred by each of the t outside home zone nodes are: $T_{\$}(v$ signed location claims $) + R_{\$}(v$ signed responses $) = T_{\$}(v$ signed 48 bit messages$) + R_{\$}(v$ signed 48 bit response messages$) = T_{\$}(48v) + R_{\$}(48v)$. For $v = 2$, we see that this yields a one-time communication expense per outside node of $T_{\$}(96) + R_{\$}(96) = 8.352$ mJ. For the v verifiers, the communication costs are identical, 8.352 mJ, by the same calculation as before.

In the above calculations, we are ignoring the costs of the routing protocol and the expense incurred by the intermediate routing nodes. Our reasoning stems from the following observation: notice that most of the genuine nodes who find themselves outside of the home zone are no more than a few hops away from their home zone. However, replicas deployed by the adversary might find themselves at any distance from their home zones. To this end, our analysis depends on the actions of the adversary to truly compute the overall communication cost. Notice that this gives the adversary an interesting denial of service attack against the sensor network, and we discuss this in Section 8.

Also of importance is to note that the communication cost is evenly distributed over all of the nodes, as they are chosen uniformly at random in the home zone and also distributed over time for a particular node. This is quite important in the sensor field to prevent some nodes from exhausting their energy prematurely, and thus, it is imperative that our protocol avoid performance peaks. Clearly, the global communication costs of one cycle grows linearly with the network size, but stays constant from the perspective of a single node. We stress that signing and verification are energy intensive operations, and our objective is to make minimal use of these operations and to discuss the costs associated with these operations below. We also note that such public key operations have been used extensively in prior work, and their costs are not accounted for on a practical level (excluding direct measurements on real sensor motes), as we do in our work.

6.3. Computation Cost

We view the computation cost as the number of public key signature and verification operations per sensor node. As discussed earlier, a sensor node deployed outside its home zone only performs a single signature generation operation in its lifetime. As computed earlier, the probability of a node to reside outside its home zone is $e^{-\frac{r^2}{2\sigma^2}}$, where r is the home zone radius and σ is the standard deviation of the Gaussian distribution. In our deployment of 50 nodes per territory, on average, about 45 nodes were deployed within the home zone and five nodes outside the home zone. Each of these t out zone nodes contact v verifiers in the home zone. The cost of signing/verification of a message m, $S_{\$}(m)$, where m is 96 bits (from our earlier calculation), is 135 mJ. Thus, each of the t out zone nodes and the v verifiers spend 135 mJ for signing operations. When these out zone nodes present their evidence to a neighboring node, they perform a verification, as well, which costs the same 135 mJ of energy. Since these are expensive operations, we ensure that this is a one-time expense in $\mathbf{Discard}$, and the costs are distributed across the nodes in the home zone and potentially across time, as well.

6.4. Storage Cost

Since sensor motes operate under limited memory, we ensure that the persistent storage costs associated with $\mathbf{Discard}$ are well within the available memory limits of current sensor motes. For example, Mica2 motes have 4 KB RAM, as listed in the data sheet by the manufacturer CrossBow Technologies [43]. In $\mathbf{Discard}$, each node maintains two caches, and we showed that the total size of these caches is $48c + 80d$ bits for neighborhood and duplicate caches of sizes c and d, respectively. For $c = 5$ and $d = 5$, we see that this size is 640 bits, well within the Mica2 data storage capability, leaving enough room for temporary working memory space for the $outsideHomeZone$ cache. Clearly, there is also enough memory available to run the sensor network protocols of primary interest and, if needed, to increase the cache sizes, as well, to improve the detection performance.

6.5. Additional Security Discussions

In this section, we discuss a couple of additional security considerations that need to be taken into account before employing $\mathbf{Discard}$ in the sensor network.

6.5.1. Need for the Secondary Epidemic

In this section, we discuss the motivation behind disseminating the identities of the duplicate nodes using the secondary epidemic. This might seem like a digression from the main thread of our discussion. However, in light of the communication cost analysis above, it makes sense to explain this here. Upon a quick examination of $\mathbf{Discard}$, it might seem that the secondary epidemic is unnecessary and clearly increases the communication costs by necessitating the exchange of the duplicate cache. We now argue in its favor and justify our design. Of course, it is true that by exchanging and merging the neighborhood caches amongst each other, nodes can identify replicas in their midst. It has been shown [42] that merely exchanging and merging the neighborhood caches among the home zone nodes creates an almost random graph G among these nodes. Here, the graph G is viewed as a directed "who-knows-whom" graph. A directed edge e between nodes n and p implies that $n.NCache$ contains p. This indicates that each node in the home zone "knows" c other random nodes in the home zone. In the course of exchanging their caches and merging, every node has been populating their duplicate caches as they find them. When an outside node requests verification from v verifiers, we would like to know how many verifiers to choose to attain sufficient confidence in the verification results. Essentially, we would like to choose v verifiers such that their union comprises the identities of all of the nodes in the entire home zone with high probability. This way, when the verification request comes in from a duplicate node, there is a high probability that at least one of the verifier nodes will identify the duplicate and trigger the appropriate revocation mechanism in place. Recall that the $NCache$ of each node is of size c and contains almost random elements from its home zone set H of size h. It has been shown that the expected size of the union of v random subsets of cardinality c drawn from a set H of size h is given by

$h\left(1-\left(1-\frac{c}{h}\right)^{v}\right)$ [44]. Suppose, $c = 5, h = 45, v = 3$; we get that this expected size is about 13. This shows that we need to increase both c and v in order to get the expected size close to 45. For instance, with a cache size of $c = 20$ and $v = 5$ verifiers, we get the expected size around 43. However, clearly, this is quite expensive in terms of communication, and hence, our decision to disseminate the duplicate caches as a secondary epidemic in **Discard** is justified.

6.5.2. Need for Multiple Verifiers

In this paragraph, we discuss the importance of using multiple verifiers in **Discard** , *i.e.*, $v > 1$. We point out the obvious fact that our scheme's robustness against the adversary increases with v, naturally with increasing communication cost. Recall that the verifier nodes are contacted by those nodes that are deployed outside their home zone, and this happens only once for each such out zone node. To understand the important role played by these verifiers, let us assume that the adversary has compromised p fraction of nodes in the home zone. When the verifier nodes are chosen at random, based on the seed broadcast by the base station, it is likely that some of these v verifiers might be under the control of the adversary. The probability of all of the v verifiers to be compromised by the adversary is clearly p^v. With probability $1 - p^v$, our scheme will be able to detect and one of the uncompromised verifier node will respond with a $Sig_p(duplicate)$ signature to the sender and take necessary revocation steps. However, if p is high, the adversary has compromised and has control over a large fraction of nodes in the home zone, then it becomes increasingly expensive, in terms of communication, to increase v to thwart such an adversary. One possible way to offset this problem is to potentially disseminate the duplicate node information of a territory across to nodes in the neighboring territories, as well. Then, the verifier nodes can be chosen from not just the home zone, but also from neighboring nodes. A second option is for the sensor network architect to increase the number of nodes in the home zone. A final complication that arises when the adversary has control over a large fraction of nodes in the home zone is this: the replicated nodes may choose to wait for a few time cycles accumulating false genuine signatures from these compromised nodes, thereby defeating **Discard** (unless they happen to accidentally contact an uncompromised node in the meantime). We raise these questions to motivate further work in this area and delegate these tasks to future work.

6.5.3. More than d Duplicates

Thus far, we have presented a scenario where the adversary compromises no more than d sensor nodes from a particular home zone and deploys as many copies of these nodes in the network. Our scheme is capable of detecting with 100% accuracy (given sufficient time) any of the duplicates deployed in the network, so long as the distinct identities of these compromised nodes from within a specific home zone do not exceed d. The adversary may duplicate a compromised node many times; however, **Discard** only saves the information of two distinct locations of every duplicate node in its cache. For instance, in the examples before, we have set $d = 5$, which is greater than 10% of the nodes in the home zone. We believe, as do researchers in prior work, that compromising more than 10% of nodes in a home zone imposes a significant effort on the adversary. Furthermore, an adversary that has no limits on the number of nodes he may capture and compromise can defeat any protocol running in the network. To this end, the size of the duplicate cache, d, should be chosen taking this effort into consideration before employing **Discard** in a sensor network. Additional considerations include the degree of the unattended nature of the network. In a large sensor field, perhaps it is not quite feasible for the adversary to compromise these many nodes, particularly in the presence of some secondary physical monitoring in place. Increasing d causes increased communication expenses, and this trade-off needs to be analyzed. However, once the adversary is able to compromise more than d nodes in a home zone, our detection accuracy drops. This is because the sensor nodes will be forced to swap out some of the duplicate node identities with information from their neighbors in the merge process. We have not explored this phenomenon and leave this task for future work, as well.

6.5.4. Revocation

As a final note, in this work, we focus on identifying only the duplicate nodes in a network. Revocation can be accomplished by using any of the schemes presented in earlier work [10,18], and we do not discuss this procedure any further in this work.

6.5.5. Putting It All Together

From the discussions thus far, we can see that **Discard** defends a sensor network against the smart/ubiquitous adversary as defined by Conti *et al.* [14,15] and the possible vectors of attack by the adversary described earlier. Furthermore, **Discard** is designed to be organic in the sense that the protocol's functionality resembles the natural operations of interactions among the sensor nodes and does not induce any undue burden on the network. It is clearly scalable and robust against the adversary. The adversary is unable to thwart **Discard** effectively, because he or she cannot *a priori* figure out which nodes in the home zone are going to serve as verifiers, unless he decides to compromise a large fraction of nodes, which is hard, as discussed earlier. The fundamental tenet of **Discard** is that it is guaranteed that the identity and location of a duplicate node will be propagated by a node once it sees it. Of course, if many nodes hold the duplicate node identity/location information, the process is accomplished even faster. In the next section, we show how **Discard** compares to existing detection schemes.

6.6. Summary of Our Results and Comparison to Prior Schemes

We present a comparison of existing schemes vs **Discard** in Table 1. This table illustrates the memory, communication and computation costs for each protocol. The communication costs are for the entire network per time period, and the memory costs are per node. Computation costs shown are per node, per time period. g stands for the number of witnesses/verifiers selected by each neighbor of a node; p is the probability a neighbor will transmit location claim to witnesses; n is the number of nodes in the sensor network; d represents the average degree of each node; and s is number of sensors in a group/cell. Since it is infeasible, due to space considerations, to describe the costs for each of these existing protocols in complete detail here, we refer the reader to the references listed in the table for a more thorough comprehension and insight into these costs. ND in Table 1 stands for "no discussion", as in the paper that presented the scheme does not discuss the cost in question.

Table 1. Comparison of existing schemes *vs.* **Discard**. ND, no discussion; SDC, single deterministic cell; P-MPC, parallel multiple probabilistic cells.

Schemes	**Communication**	**Storage**	**Computation**
Broadcast [10]	O(n^2)	O(d)	ND
RM [10]	O(n^2)	O($\sqrt{n}$)	ND
LSM [10]	O($n\sqrt{n}$)	O($\sqrt{n}$)	ND
SET [17]	O($n\sqrt{n}$)	O(n)	O(1)
RED [14,15]	O(gpd)	O(gpd)	O(gpd)
SDC [16]	O($ps\sqrt{n}$)	O(1)	ND
P-MPC [16]	O($ps\sqrt{n}$)	O(1)	ND
Scheme III [18]	O(n)	O(s)	O(1)
CSI [45]	O($n \log n$)	ND	ND
Discard	O(n)	O(1)	O(1)

In summary, we highlight the most notable features of **Discard** as a list below:

- Scheme III by Ho *et al.* [18] and similar duplicate detecting schemes do not seem to place an emphasis on detecting duplicates within a small geographic region (say, within a home zone). The reasoning is that the adversary can exert his or her influence only within this small region, which is reasonable. However, in our scheme, we detect these duplicates, as well.
- We present an explicit relationship between storage costs and detection accuracy.
- It is also noteworthy that, in **Discard**, the detection rate is independent of the deployment accuracy. However, as expected, the communication cost of course depends on the deployment accuracy.
- As noted earlier, since every location claim is digitally signed, the adversary cannot make uncompromised nodes appear to be replicas by faking their location claims. Thus, **Discard** works without any false positives.
- The schemes we outlined in prior work, Section 2, use broadcast mechanisms to propagate duplicate node information in the home zone. However, these broadcast communication costs are typically not accounted for and are deemed negligible. In our scheme, we account for all costs explicitly.
- Lastly, unlike Scheme III by Ho *et al.* [18], our storage costs are independent of the group size.
- In this paragraph, we sketch a quick comparison of our work with the scheme called CSI [45], depicted in the table above. In the protocol CSI, presented by Yu *et al.* [45], the detection of duplicate nodes in the sensor network is accomplished by the scheme CSI that employs a novel signal processing technique called compressed sensing. CSI has a communication cost of O($n \log n$), which the authors show to be the lowest possible communication cost in an unstructured sensor network. However, in our work, we are dealing with a group-deployed static sensor network. To this end, we are able to enjoy a lower communication cost of O(n). Naturally, these two schemes are not directly comparable because of the differences in their underlying network topologies. Furthermore, while the authors demonstrate that CSI has low

communication costs, the discussion of computation and storage costs incurred by the scheme for each time period is missing and has not been discussed in the work. Since CSI uses "TAG: a Tiny AGgregationService for Ad-Hoc Sensor Networks" [46] and builds an aggregation tree rooted at the base station among the sensor nodes to accomplish its duplicate detection mechanism, we argue that the storage and computation costs are proportional to the tree generation and maintenance subroutines. This also imposes the requirement that the sensor nodes have been scheduled such that the data can be aggregated along the tree "level-by-level" at each time instant. In summary, while CSI does accomplish duplicate detection efficiently, it does so with the help of underlying subroutines (whose costs are not accounted for) and with a special purpose tree topology, which may or may not be conducive in all deployment models. The same is true of our scheme, as well. **Discard** also employs a group-deployed structure, and one might argue that this deployment model is not suitable in all circumstances.

7. Implementation Details

In this paper, we have analyzed **Discard** primarily via simulation. While we have analyzed our algorithm analytically and demonstrated some of its properties quantitatively, a large fraction of our evaluation has been done using simulated experiments. We implemented our simulation framework using the Peersim discrete event-based simulator implemented in Java [47]. We chose Peersim because it is a peer-evaluated, open source, general purpose discrete event simulation environment.

8. Future Work and Discussion

In this section, we highlight a couple of aspects of **Discard** that can be improved to achieve better performance guarantees. Since we assume that nodes were deployed in groups, we were able to save on communication, computation and storage costs dramatically over earlier work, as we do not need to generate location claims for every interaction among nodes. There is another vector of information that may be utilized to further reduce these costs. If we assume that there is a notion of time synchronization among the sensors, we may utilize this vector to determine the legitimacy of nodes, which may obviate the need for some verification messages, leading to further energy savings. Verification may be required only when both the time and space information for a node are amiss. An additional improvement that may lead to further communication cost savings is: before exchanging the neighborhood or duplicate caches, nodes might consider computing the checksum and, if different, decide to exchange and merge the caches. However, this necessitates storing the caches in a specific order and introduces computational costs.

In addition to some of the questions that were raised in Section 6.5, observe that **Discard** , like most other replica detection schemes, is susceptible to a denial of service attack by the adversary. However, **Discard** is more resilient to this attack than existing schemes outlined in Section 2 of prior work. First, observe that our scheme places the burden of collecting the evidence signatures from the verifier nodes on the nodes seeking authentication. Suppose an adversary were to deploy duplicate nodes, claim falsified locations and contact the verifier nodes. While this does deplete the energy of the verifiers who are distributed randomly across time and the home zone, it does so much more rapidly on the duplicate nodes themselves. Verifiers do not process any verification messages unless they are indeed chosen to be verifiers by the base station random seed. Hence, the denial of service attack that the adversary has launched is expected to be short-lived. However, a more satisfactory solution would be desirable.

On another note, observe that when a node n determines the IDs of the v verifier nodes from its home zone to acquire verification evidence, it is possible (with a small probability) that the node IDs generated, using the random seed broadcast by the base station, are themselves nodes deployed outside the home zone. In this case, node n will not receive the response it expects. In this case, node n collects the signatures it receives and is forced to wait another time period to contact more verifiers to reach the threshold of v signatures to be accepted into the network before participating in network activities.

In the future, we would like to test **Discard** 's capabilities on diverse topologies with varying deployment accuracy. While, theoretically, we expect our epidemic techniques to yield similar results, it would be interesting to observe practical consequences of diverse topologies and poor deployment accuracy.

Our motivation to design **Discard** in its present form was to keep it simple and lead to easy implementation. Furthermore, **Discard** requires very few guarantees from the underlying communication platform, which is not true in some of the prior schemes. Some other fundamental exploratory questions to consider, at the expense of adding complexity, are: the choice of peer sensor nodes and the choice of verifier nodes. We would like to study the potential impact of the "power of two random choices" [48], a simple, yet powerful construct that shows that a small amount of choice can yield significant performance improvements in a randomized algorithm.

Lastly, we would like to build discard using the other variants of epidemic algorithms, namely a combination of anti-entropy, direct mail and rumor mongering, while exploring the option of having them propagate in the network at different time periods while maintaining a fine-grained control over the spread of each of the epidemics. Furthermore, it is important, practically, to take collisions and communication interference into account when deploying **Discard** in a real-world sensor network.

9. Conclusions

In this paper, we presented **Discard** , a distributed, randomized, epidemic-based duplicate detection algorithm. We showed that **Discard** has a 100% detection accuracy with modest communication, computation and storage overheads in comparison to existing schemes in the literature. Furthermore, we note that, to the best of our knowledge, ours is the first scheme to employ epidemic, gossip-inspired techniques to detect duplicates in a sensor network. We have analyzed **Discard** both analytically and via empirical simulation experiments. We conclude that epidemic algorithms have a strong potential and should be explored further in this arena. While similar techniques have been actively employed in other areas of distributed computing, we were surprised that they have not been utilized to their full potential in wireless sensor networks. To this end, we hope that our work inspires further research along these lines.

Acknowledgments

The first author would like to thank the support offered by SHSUGrant ERG456914. The authors are solely responsible for the views expressed in this paper, which do not necessarily reflect the position of the supporting organizations. The authors gratefully acknowledge the comments of the reviewers who helped improve the quality of the manuscript significantly.

Author Contributions

All authors contributed equally to this work.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Chan, H.; Perrig, A.; Song, D. Random key predistribution schemes for sensor networks. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 11–14 May 2003; pp. 197–213.
2. Eschenauer, L.; Gligor, V.D. A key-management scheme for distributed sensor networks. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, 18–22 November 2002; pp. 41–47.
3. Akyildiz, I.F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. A survey on sensor networks. *IEEE Commun. Mag.* **2002**, *40*, 102–114.
4. Akyildiz, I.F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless sensor networks: A survey. *Comput. Netw.* **2002**, *38*, 393–422.
5. Newsome, J.; Shi, E.; Song, D.; Perrig, A. The Sybil attack in sensor networks: Analysis & defenses. In Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks, Berkeley, CA, USA, 26–27 April 2004; pp. 259–268.
6. Douceur, J.R. The Sybil attack. In *Peer-to-Peer Systems*; Springer: Heidelberg, Germany, 2002; pp. 251–260.
7. Conti, M.; di Pietro, R.; Mancini, L.V.; Mei, A. Emergent properties: Detection of the node-capture attack in mobile wireless sensor networks. In Proceedings of the First ACM Conference on Wireless Network Security, Alexandria, VA, USA, 31 March–2 April 2008; pp. 214–219.
8. Shashidhar, N.; Kari, C.; Verma, R. Epidemic Node Replica Detection in Sensor Networks. In Proceedings of the Third IEEE ASE International Conference on Cyber Security (CyberSecurity 2014), Stanford, CA, USA, 27–31 May 2014.
9. Brooks, R.; Govindaraju, P.; Pirretti, M.; Vijaykrishnan, N.; Kandemir, M.T. On the detection of clones in sensor networks using random key predistribution. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2007**, *37*, 1246–1258.

10. Parno, B.; Perrig, A.; Gligor, V. Distributed detection of node replication attacks in sensor networks. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 8–11 May 2005; pp. 49–63.
11. Xing, K.; Liu, F.; Cheng, X.; Du, D.H. Real-time detection of clone attacks in wireless sensor networks. In Proceedings of the The 28th International Conference on IEEE Distributed Computing Systems, ICDCS'08, Beijing, China, 17–20 June 2008; pp. 3–10.
12. Zhu, W.T.; Zhou, J.; Deng, R.H.; Bao, F. Detecting node replication attacks in wireless sensor networks: A survey. *J. Netw. Comput. Appl.* **2012**, *35*, 1022–1034.
13. Gligor, V. Security of emergent properties in ad-hoc networks (transcript of discussion). *Security Protocols*; Springer: Heidelberg, Germany, 2006; pp. 256–266.
14. Conti, M.; di Pietro, R.; Mancini, L.V.; Mei, A. A randomized, efficient, and distributed protocol for the detection of node replication attacks in wireless sensor networks. In Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Montreal, QC, Canada; 9–14 September 2007; pp. 80–89.
15. Conti, M.; di Pietro, R.; Mancini, L.V.; Mei, A. Distributed detection of clone attacks in wireless sensor networks. *IEEE Trans. Dependable Secur. Comput.* **2011**, *8*, 685–698.
16. Zhu, B.; Addada, V.G.K.; Setia, S.; Jajodia, S.; Roy, S. Efficient distributed detection of node replication attacks in sensor networks. In Proceedings of the Twenty-Third Annual Computer Security Applications Conference, ACSAC 2007, Miami Beach, FL, USA, 10–14 December 2007; pp. 257–267.
17. Choi, H.; Zhu, S.; la Porta, T.F. SET: Detecting node clones in sensor networks. In Proceedings of the Third International Conference on Security and Privacy in Communications Networks and the Workshops, SecureComm 2007, Nice, France, 17–21 September 2007; pp. 341–350.
18. Ho, J.W.; Liu, D.; Wright, M.; Das, S.K. Distributed detection of replica node attacks with group deployment knowledge in wireless sensor networks. *Ad Hoc Netw.* **2009**, *7*, 1476–1488.
19. Liu, D.; Ning, P.; Du, W. Group-based key predistribution for wireless sensor networks. *ACM Trans. Sens. Netw. (TOSN)* **2008**, *4*, doi:10.1145/1340771.1340777.
20. Du, W.; Wang, R.; Ning, P. An efficient scheme for authenticating public keys in sensor networks. In Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Chicago, IL, USA, 25–28 May 2005; pp. 58–67.
21. Du, W.; Fang, L.; Peng, N. Lad: Localization anomaly detection for wireless sensor networks. *J. Parallel Distrib. Comput.* **2006**, *66*, 874–886.
22. Du, W.; Deng, J.; Han, Y.S.; Chen, S.; Varshney, P.K. A key management scheme for wireless sensor networks using deployment knowledge. In Proceedings of the Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004, Hong Kong, China, 7–11 March 2004; Volume 1.
23. Delgosha, F.; Fekri, F. Threshold key-establishment in distributed sensor networks using a multivariate scheme. In Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM 2006, Barcelona, Spain, 23–29 April 2006; pp. 1–12.

24. Liu, D.; Ning, P. Establishing pairwise keys in distributed sensor networks. In Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington, DC, USA, 27–30 October 2003; pp. 52–61.
25. Zhang, W.; Tran, M.; Zhu, S.; Cao, G. A random perturbation-based scheme for pairwise key establishment in sensor networks. In Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Montreal, QC, Canada, 9–14 September 2007; pp. 90–99.
26. Leon-Garcia, A. *Probability and Random Processes for Electrical Engineering*; Addison-Wesley: Reading, UK, 1994; Volume 2.
27. Albano, M.; Chessa, S.; Nidito, F.; Pelagatti, S. Dealing with nonuniformity in data centric storage for wireless sensor networks. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 1398–1406.
28. Gupta, V.; Wurm, M.; Zhu, Y.; Millard, M.; Fung, S.; Gura, N.; Eberle, H.; Shantz, S.C. Sizzle: A standards-based end-to-end security architecture for the embedded internet. *Pervasive Mob. Comput.* **2005**, *1*, 425–445.
29. Liu, A.; Ning, P. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In Proceedings of the International Conference on Information Processing in Sensor Networks, IPSN'08, St. Louis, MO, USA, 22–24 April 2008; pp. 245–256.
30. Wang, H.; Sheng, B.; Tan, C.C.; Li, Q. Comparing symmetric-key and public-key based security schemes in sensor networks: A case study of user access control. In Proceedings of the 28th International Conference on Distributed Computing Systems, ICDCS'08, Beijing, China, 17–20 June 2008; pp. 11–18.
31. Liu, D.; Ning, P.; Du, W.K. Attack-resistant location estimation in sensor networks. In Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, Los Angeles, CA, USA, 25–27 April 2005; p. 13.
32. Li, Z.; Trappe, W.; Zhang, Y.; Nath, B. Robust statistical methods for securing wireless localization in sensor networks. In Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005, Los Angeles, CA, USA, 25–27 April 2005; pp. 91–98.
33. Capkun, S.; Hubaux, J.P. Secure positioning in wireless networks. *IEEE J. Sel. Areas Commun.* **2006**, *24*, 221–232.
34. Seshadri, A.; Perrig, A.; van Doorn, L.; Khosla, P. Swatt: Software-based attestation for embedded devices. In Proceedings of the 2004 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 9–12 May 2004; pp. 272–282.
35. Demers, A.; Greene, D.; Hauser, C.; Irish, W.; Larson, J.; Shenker, S.; Sturgis, H.; Swinehart, D.; Terry, D. Epidemic algorithms for replicated database maintenance. In Proceedings of the Sixth Annual Acm Symposium on Principles of Distributed Computing, Vancouver, BC, Canada, 10–12 August 1987; pp. 1–12.
36. Eugster, P.T.; Guerraoui, R.; Kermarrec, A.M.; Massoulié, L. From epidemics to distributed computing. *IEEE Comput.* **2004**, *37*, 60–67.
37. Bailey, N.T. *The Mathematical Theory of Infectious Diseases and Its Applications*; Hafner Press/ MacMillian Pub. Co.: New York, NY, USA, 1975.

38. Albano, M.; Gao, J. In-network coding for resilient sensor data storage and efficient data mule collection. In *Algorithms for Sensor Systems*; Springer: Heidelberg, Germany, 2010; pp. 105–117.
39. Xu, Y.; Heidemann, J.; Estrin, D. Geography-informed energy conservation for ad hoc routing. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, Rome, Italy, 16–21 July 2001; pp. 70–84.
40. Wander, A.S.; Gura, N.; Eberle, H.; Gupta, V.; Shantz, S.C. Energy analysis of public-key cryptography for wireless sensor networks. In Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications, PerCom 2005, Kauai Island, HI, USA, 8–12 March 2005; pp. 324–328.
41. Eugster, P.T.; Guerraoui, R.; Handurukande, S.B.; Kouznetsov, P.; Kermarrec, A.M. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst. (TOCS)* **2003**, *21*, 341–374.
42. Jelasity, M.; Kowalczyk, W.; van Steen, M. *Newscast Computing*; Technical Report IR-CS-006; Vrije Universiteit Amsterdam, Department of Computer Science: Amsterdam, The Netherlands, 2003.
43. DataSheet, M. *Crossbow Technology*; Document Part Number: 6020-0042-08 Rev A; 2013.
44. Barot, M.; de la Peña, J.A. Estimating the size of a union of random subsets of fixed cardinality. *Elem. Math.* **2001**, *56*, 163–169.
45. Yu, C.M.; Lu, C.S.; Kuo, S.Y. CSI: Compressed sensing-based clone identification in sensor networks. In Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), Lugano, Switzerland, 19–23 March 2012; pp. 290–295.
46. Madden, S.; Franklin, M.J.; Hellerstein, J.M.; Hong, W. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Oper. Syst. Rev.* **2002**, *36*, 131–146.
47. Montresor, A.; Jelasity, M. PeerSim: A Scalable P2P Simulator. In Proceedings of the 9th International Conference on Peer-to-Peer (P2P'09), Seattle, WA, USA, 9–11 September 2009; pp. 99–100.
48. Richa, A.W.; Mitzenmacher, M.; Sitaraman, R. The power of two random choices: A survey of techniques and results. *Comb. Optim.* **2001**, *9*, 255–304.