MDSClone: Multidimensional Scaling Aided Clone Detection in Internet of Things

Po-Yen Lee, Chia-Mu Yu, Tooska Dargahi, Mauro Conti, *IEEE Senior Member*, and Giuseppe Bianchi

***Abstract*—Cloning is a very serious threat in the Internet of Things (IoT), owing to the simplicity for an attacker to gather configuration and authentication credentials from a non-tamper-proof node, and replicate it in the network. In this paper, we propose MDSClone, a novel clone detection method based on multidimensional scaling (MDS). MDSClone appears to be very well suited to IoT scenarios, as it (i) detects clones without the need to know the geographical positions of nodes, and (ii) unlike prior methods, it can be applied to *hybrid* networks that comprise both static and mobile nodes, for which no mobility pattern may be assumed a priori. Moreover, a further advantage of MDSClone is that (iii) the core part of the detection algorithm can be parallelized, resulting in an acceleration of the whole detection mechanism. Our thorough analytical and experimental evaluations demonstrate that MDSClone can achieve a 100% clone detection probability. Moreover, we propose several modifications to the original MDS calculation, which lead to over a 75% speed up in large scale scenarios. The demonstrated efficiency of MDSClone proves that it is a promising method towards a practical clone detection design in IoT.**

I. Introduction

Internet of Things (IoT) is an emerging networking paradigm, in which a large number of interconnected devices communicate with each other to facilitate communications between people and objects [1]. For example, a smart city is composed of several smart sectors, such as [2] smart homes, smart hospitals, and smart cars, which are significant applications of IoT. In a smart home scenario, each IoT gadget is equipped with embedded sensors and wireless communication capabilities. The sensors are able to gather environmental information and communicate with each other, as well as the house owner and a central monitoring system. In a smart hospital scenario, which could be implemented using body sensor networks (BSN), patients wear implantable sensors that collect body signals and send the data to a local or remote database for further analysis. As another example, in a smart traffic scenario embedded sensors in cars are able to detect accident events or traffic information, and collaboratively exchange such information.

On account of their restricted features and capabilities, IoT devices are vulnerable to several security threats [3]. For example, IoT devices could easily be captured, leading to a *clone attack* (also known as a *node replication attack*). In such a scenario, the captured device is reprogrammed, cloned, and placed back in the network. Moreover, in special cases (e.g., misconfiguration or production by untrusted manufacturers with adversarial intentions) devices that are supposed to be trusted can cause clone attacks [4]. A clone attack is extremely harmful, because the clones with legitimate credentials will be considered as legitimate devices. Therefore, such clones can easily perform various malicious activities in the network [5], [6], such as launching an insider attack (*e.g., blackhole attack*) and injecting false data leading to hazards in an IoT scenario.

P.-Y. Lee is with the Yuan Ze University, Taiwan.
C.-M. Yu is with the National Chung Hsing University and Taiwan Information Security Center (TWISC@NCHU), Taiwan.
T. Dargahi is with CNIT - University of Rome Tor Vergata, Italy, and University of Salford, Manchester, UK.
M. Conti is with the University of Padua, Italy.
G. Bianchi is with CNIT - University of Rome Tor Vergata, Italy.

Problem Statement. While there exists fairly extensive literature on clone attack detection approaches in WSNs [7], [8], this remains an open problem when it comes to IoT scenarios. In particular, compared with conventional WSNs, two unique characteristics of IoT environment make the establishment of clone detection schemes in IoT a more challenging issue. First, there is a lack of accurate geographical position information for the devices. For instance, the devices embedded in smart cars are likely to derive their location information via the car navigation system, i.e., geographical positioning system (GPS), while the devices in a smart home or BSN are unlikely to have embedded GPS capability, owing to its high energy consumption and extra hardware requirements [9]. Second, IoT networks are hybrid networks composed of both static and mobile devices without a priori mobility pattern (they can be static or moving with high or low velocity) [10], e.g., a patient carrying wearable sensors and living in a smart home. Wearable devices could be considered as mobile nodes, because the patient may move around, while most of the devices in a smart home are immobile. In fact, IoT nodes are relocatable, without an a priori mobility pattern (they can be static, moving with high velocity, or moving slowly) [10]. Although some of the existing clone detection methods for mobile networks (e.g., [11]–[13]) could be applied to hybrid networks (composed of both stationary and mobile devices), these suffer from a certain detection probability degradation. In what follows, we explain how we address these challenges and advance the state-of-the-art solutions in detecting clone attacks.

Contribution. In this paper, we propose MDSClone, a novel clone detection mechanism for IoT environments. MDSClone specifically circumvents the two major above-mentioned issues that emerge in IoT scenarios by adopting a *multidimensional scaling* (MDS) algorithm [14], [15]. In particular, our main contributions are as follows.

1) We propose a clone detection method that does not rely on geographic positions of nodes. Instead, by adopting the MDS algorithm, we generate the network map based on the relative neighbor-distance information of the nodes. While most of the state-of-the-art clone detection methods assume that each node is always aware of its geographical position, this assumption does not hold for all the IoT devices [9]. Therefore, by removing such an assumption in MDSClone, we significantly advance the existing clone detection solutions for IoT.
2) Our proposed MDSClone method is capable of detecting clones in the network based on topology distortion, without considering any specific mobility pattern. This is

an important feature of MDSClone, since as explained earlier, IoT nodes do not follow a particular mobility pattern, and existing clone detection methods for mobile networks do not have reasonable performance in hybrid networks (for more details please refer to Section II). Compared to the related work, MDSClone method is applicable for all pure static, pure mobile, and hybrid networks, and the detection probability of MDSClone remains the same for all of these network topologies.

3) We show that MDSClone is efficient in terms of the computational overhead, because the main computation is performed by the base station (BS), and the server-side computation can easily be parallelized to significantly improve the performance. This is an outstanding feature of MDSClone compared to the state-of-the-art, as the parallelization capability of the existing clone detection methods remains unclear.
4) Along with the main MDSClone algorithm, we also propose three techniques (i.e., *CIPMLO*, *TI*, and *SMEBM*) to speed up the core part of MDSClone, which comprises the MDS calculation.
5) We provide a thorough evaluation of our proposed method considering different evaluation criteria, i.e., the clone detection probability and computation time of our algorithm when adopting our proposed speed-up methods. Moreover, we provide analytical and experimental comparisons of MDSClone with state-of-the-art clone detection methods. Our reported experimental results exhibit a perfect detection of clone nodes in the network, requiring a constant amount of memory and a reasonable communication overhead.

II. Related Work

In recent years, owing to the increasing interest in adopting WSNs in several applications, there has been a surge of interest in providing WSN-specific security solutions, amongst which clone attack detection has attracted significant attention. In this section, we review the clone detection methods that are most closely related to our work, and clarify the difference between our proposal and the existing related work.

Researchers [7], [8] have proposed several classifications for clone detection approaches based on the required information (i.e., location-based or location-independent), detection methods (i.e., centralized, distributed, or partially distributed), and supporting network type (i.e., mobile or static networks). Our proposed MDSClone approach falls in the category of location-independent centralized methods supporting hybrid (both static and mobile) networks. We believe that the centralized nature of MDSClone is not a drawback, considering the emerging municipality-scale IoT networking technologies such as NarrowBand-Internet of Things (NB-IoT) [16] and LoRaWAN [17]. Indeed, a centralized security monitoring solution is perfectly inline with the hierarchical architecture fostered by such technologies, which are currently being supported by key players, including among others Cisco and Orange. For instance, the current LoRaWAN deployment being developed in the city of Rome concentrates all IoT sensor traffic collected by several tens of radio stations spread across the whole of the Rome municipality and relevant neighbors in a (logically) single centralized network server, which therefore appears to be a natural candidate to further host anomaly detection approaches such as MDSClone.

In the case of static networks, a popular approach for detecting clones is *witness finding*. In essence, the idea behind witness finding is that the existence of clones must lead to location conflicts. More specifically, each node u collects the location information, $L(v)$, of its neighboring nodes, e.g., v, and sends the collected *location claims* $\langle v, L(v)\rangle$ to some selected nodes. Nodes receiving two location claims with the same ID v, but with two distinct locations, will serve as *witness nodes*, and witness the location conflict. The witness finding strategy not only detects the existence of clones, but also identifies the clone IDs.

A network-wide broadcast is the simplest way to find a witness, but this incurs a prohibitive communication cost. In [18], the authors proposed two approaches, randomized multicast (RM) and line-selected multicast (LSM), in order to reduce the communication costs of network-wide broadcasts. Two other approaches proposed in [19], i.e., single deterministic cell (SDC) and parallel multiple probabilistic cells (P-MPC), share the same spirit as RM and LSM. However, SDC and P-MPC are only efficient when the network is partitioned into cells. Compared with the aforementioned approaches, the protocol proposed in [20], i.e., the randomized, efficient, and distributed (RED) protocol, provides an almost-perfect guarantee of clone detection. RED utilizes a special centralized broadcasting device, such as a satellite and UAV, in order to periodically broadcast the node IDs responsible for detecting particular conflicting location claims. In another study, Zhang et al. [21] proposed four clone detection methods that take advantage of double ruling and the Bloom filter. Recently, Dong et al. [22] proposed the low-storage clone detection (LSCD) method, taking into account the memory requirements and residual energies of nodes. An inherent weakness among all of the witness finding-based approaches is the assumption of the knowledge of location information available for each node. A couple of solutions take alternative approaches to detect clones, such as the social fingerprint [23], predistributed keys [24], and random clustering [25] methods.

In the case of mobile sensor networks, by using a simple challenge-and-response strategy, XED [11] presents the first distributed clone detection method for mobile networks. However, it is vulnerable to collusions of the cloned nodes. EDD [11], [12] is a distributed clone detection method based on the discrepancy between the distributions of the numbers of encounters with clone and ordinary nodes. In [26], a base station (BS) collects the geographical positions of nodes, looking for a clone moving with a speed exceeding the pre-configured speed limit. In [5], [13], the same idea is employed, but the ordinary nodes play the role filled by the BS in [26].

We argue that, although most existing clone detection methods proposed for mobile networks could be applied to hybrid networks as well, this adoption will degrade the security and clone detection probability. The clone detection methods for mobile networks that do not (fully) rely on velocity violations include XED [11], EDD [11], [12], TDD [13], SDD-LC [13], SDD-LWC [13], and HIP-HOP [5]. The reason that XED and EDD suffer from a security degradation when applied to hybrid networks is that clones that are aware of the positions of static nodes can either choose not to enter the proximity of static nodes, or to enter at certain time slots. If so, static nodes in XED do not have a chance to exchange secret information with different clones. Moreover, in EDD the number of times that each static node will encounter a clone

node will be controlled by clones. Therefore, if clones adopt the above evasion strategy, then only the mobile nodes (rather than both static and mobile nodes) will be able to detect clones, reducing the probability of clone detection. On the other hand, the detection effectiveness of TDD, SDD-LC, and SDD-LWC partly relies on whether each node encounters a particular node too many times (similarly to EDD). As a consequence, if the clones adopt the above-mentioned evasion strategy, then the detection capabilities of the TDD, SDD-LC, and SDD-LWC methods will also be degraded. In addition, the HIP-HOP approach detects clones based on the fact that if two witness nodes are either one-hop or two-hop neighbors, then either the witness nodes or the node connecting two witness nodes will find the location conflict of clones. However, if witness nodes far away from each other happen to both be static, then they have no chance of being either one-hop or two-hop neighbors, thus reducing the probability of clone detection.

In summary, the existing clone detection methods devised for static networks cannot be applied to scenarios where node mobility would destroy the neighborhood and distance relations among the nodes. On the other hand, as mentioned above, the adoption of most of the mobile clone detection methods to hybrid networks results in a degradation of the clone detection probability. Therefore, in order to deal with clones in IoT environments, we need to provide a method that is "particularly" designed for hybrid networks, and does not rely on any assumption regarding the mobility pattern, if any. In addition, prior solutions largely rely on the assumption that each node is aware of its geographical position. However, as explained in Section I, this is not the case for IoT devices. As a consequence, the existing clone detection methods are not applicable to IoT environments. Table I presents a comparison between MDSClone and the other existing clone detection schemes, in terms of the communication and memory overhead, required information, and network type.

III. System Model

In this section, we describe our considered network model and assumptions (Section III-A), as well as the attack model (Section III-B). Table II presents the list of employed notations.

A. Network Model

We consider an IoT network as a hybrid network consisting of two main entities: 1) n static and mobile nodes with unique IDs [29]: ID $\in \{1, \ldots, n\}$; and 2) a base station (BS). Each IoT device periodically measures its distance with its neighboring nodes, and sends the information to the BS. In our system model, the BS is in charge of executing our proposed MDSClone algorithm and locating the "clones" (for a definition please refer to Section III-B) in the network. In particular, the BS periodically receives neighboring information for each node in the network, and constructs a location map (based *only* on the information received from the nodes) in order to detect clones (we explain the details of the MDSClone algorithm in Section V-A). The BS executes MDSClone offline, and each generated location map is dedicated to a snapshot of the network at time t. The main idea in our proposed method is that at time t, a node x cannot have two different sets of neighbors, which means that x cannot be in two different locations of the network at time t. In our network model, we make the following assumptions:

- We assume that nodes are not "necessarily" aware of their exact geographical position. This assumption is based on the following two factors explained in the existing literature: i) As explained in [9], using GPS is costly in terms of energy and the requirements for extra hardware, and ii) researchers [30] believe that GPS-based positioning is not efficient in indoor scenarios. Therefore, we assume that some nodes (e.g., smartphones) may be GPS-enabled, and others (e.g., home appliances) may not. Hence, our proposed method does not rely on geographical positions of nodes. This assumption is to address the first challenge that we mentioned in the "Problem Statement" Section, i.e., lack of accurate geographical position information of the devices.
- We assume that mobile nodes are moving without any particular mobility pattern. This assumption makes our network model more realistic, because the mobility patterns of nodes (e.g., wearable sensors) in IoT scenarios are unpredictable, as explained in [10]. We make this assumption to consider the second challenge that we mentioned in the the "Problem Statement" Section, i.e., IoT networks are hybrid networks composed of both static and mobile devices without a priori mobility pattern.
- We also assume that IoT devices are capable of enacting short-range device-to-device communication (as explained in [10]). Therefore, each node can measure its distance from its neighboring nodes via radio signal strength (RSS) or time of arrival (ToA) (as comprehensively discussed in [9], [30]). Although the estimated distances are not perfectly accurate, they are sufficient for our approach. We make this assumption, as in our proposed approach, each IoT device should periodically measure its distance with its neighboring nodes and send to the BS.
- We assume that the BS knows the geographic positions of IoT devices at the very beginning (only during the initialization of the network). However, after the network deployment, the BS is no longer aware of the positions of the devices. We make this assumption because the setup and deployment of IoT devices in the network are generally performed by the network designer, and so it is reasonable to adopt such an assumption. This assumption helps the BS in detecting and locating the clone nodes by comparing the constructed location map by the information received from the nodes and the original network map.
- We also assume that there exists a loose time synchronization between the nodes[1], and the network operation time is divided into time intervals, each of which has the same length. These assumptions are in line with other clone detection methods, e.g., [5], [31]. We make this assumption since each generated location map is dedicated to a snapshot of the network at time t.
- We assume that the exchanged messages are digitally signed[2] before being sent out, unless stated otherwise. We have studied the practicality and efficiency of such operations in [32], [33]. We make this assumption to ensure the confidentiality and accuracy of the exchanged neighboring information, based on which the location

[1]Because IoT devices are usually assumed to have an Internet connection, relying on the network time protocol (NTP) could be one solution to achieve a loose time synchronization among nodes.

[2]Similar to [18], [31], we can assume that a light weight ID-based public key cryptosystem can be used by the nodes.

TABLE I: Comparison between different clone detection schemes.

Schemes	Communication cost	Memory cost	Location-based	Network type
RM [18]	$O(n^2)$	$O(\sqrt{n})$	Yes	Static
LSM [18]	$O(n\sqrt{n})$	$O(\sqrt{n})$	Yes	Static
Social Finerprint [23]	$O(n\sqrt{n})$	$O(1)$	No	Static
CC-MEM [21]	$O(n\sqrt{n})$	N/A	Yes	Static
RAWL & TRAWL [27]	$O(n\sqrt{n}\log n)$	$O(\sqrt{n}\log n)$	Yes	Static
SDC & P-MPC [19]	$O(ndp\sqrt{n}) + O(s)$	$O(t)$	Yes	Static
RED [20]	$O(nwdp\sqrt{n})$	$O(wdp)$	Yes	Static
ERCD [28]	$O(n\sqrt{n})$	$O(1)$	Yes	Static
LSCD [22]	$O(n\sqrt{n})$	$O(\lceil l/r \rceil)$	Yes	Static
TDD [13]	$O(\sqrt{n})$	$O(n)$	Yes	Mobile
SDD-LC & SDD-LWC [13]	$O(1)$	$O(n)$	Yes	Mobile
SPRT [26]	$O(\sqrt{n})$	$O(1)$	Yes	Mobile
XED [11]	$O(1)$	$O(n)$	No	Mobile
EDD [11]	$O(1)$	$O(1)$	Yes	Mobile
HIP-HOP [5]	$O(n)$	$O(1)$	Yes	Mobile
MDSClone (proposed approach)	$O(n\sqrt{n})$	$O(1)$	No	Hybrid

n: number of nodes, w: number of witnesses generated by a neighboring node, d: average node degree, p: probability of forwarding a message, t: number of witness nodes that store a location claim, c: number of nodes in a cell, l: length of a witness path, r: transmission range of a node.

TABLE II: List of notations used in this paper.

Notation	Description
BS	Base station.
n	Number of nodes in the network.
$d_{i,j}$	Distance between nodes i and j.
$D \in \mathbb{R}^{n\times n}$	Distance matrix (distance between each pair of nodes).
$X \in \mathbb{R}^{n\times p}$	Coordinate matrix (ground truth node map).
$B \in \mathbb{R}^{n\times n}$	Inner product matrix.
$X'_t \in \mathbb{R}^{n\times p}$	Reconstructed coordinate matrix (node map).
λ	Distortion threshold.
$\mathcal{L}_t$	Neighbor-distance information received by the BS at t-th time interval.
$\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \bar{\mathcal{L}}_t)$	Distortion function.
$\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$	Localization function.
ℓ_i^t	Location of node i at time interval t (a column vector).

map will be generated.

It is worth mentioning that all of our assumptions are consistent with the existing literature, and there are several real-world applications supporting our assumptions. Examples of IoT networks are the smart home and smart city, where a large number of static and mobile devices with unique IDs collaborate to provide a better quality of life for humans. For example, the Samsung SmartThings Home Monitoring Kit[3] provides a hybrid network in which several static nodes (e.g., smart fridges, smart lamps, or smart thermostats) and mobile nodes (e.g., smartphones) can connect to each other. This kit comes with a hub and several smart things that could connect to each other and the hub through single-hop or multi-hop communication (using the repeaters that exist in the kit)[4]. Another example is the Cisco Smart City[5], in which there are static nodes (e.g., traffic lights) and mobile nodes (e.g., Internet-connected cars). Moreover, several EU projects (e.g., the EU H2020 Wise-IoT project[6] or Santander city in the UK[7]) are examples of real-world heterogeneous IoT environments.

[3]https://www.youtube.com/watch?v=XQfAqlc7Vj8
[4]https://blog.smartthings.com/iot101/a-guide-to-wireless-range-repeaters/
[5]https://www.youtube.com/watch?v=x6WfZlETbx4
[6]http://wise-iot.eu/en/home/
[7]http://www.smartsantander.eu/

B. Attack Model

IoT devices are usually considered not to be tamper-resistant [34]. In other words, the stored security credentials can all be extracted in the case of a device bein compromised. Moreover, the adversary can compromise a device immediately after the node deployment. No secure bootstrapping time is available. Thus, the adversary can access all of the legitimate credentials of the compromised devices. In this paper, we consider an adversary that is capable of performing "clone attack", meaning that they are able to fabricate compromised devices and store the legitimate credentials from the compromised devices inside several fabricated devices, which is (consistent with related work on clone detection such as [11]). A compromised node, as well as the fabricated nodes that have the same ID and credentials as the compromised node, are called *clones*. Clones can communicate and collude with each other, attempting to subvert the detection functionality in a stealthy manner. It should be noted that we only consider cloning attacks, and we assume there is no concurrent "node compromise" attack, meaning that no other nodes (beyond the clones) act in a malicious manner.

In particular, we deal with clone attacks and *not* single node compromise attacks. In essence, a "clone attack" can be considered as a special kind of node compromise attack, in which there are two or more compromised nodes with the same ID in the network at the same time. In other words, clone nodes are exact copies of the original compromised node. Although the first step of conducting a clone attack is to compromise a single node, we only consider the aftermath of compromising and cloning (similar to all the related work in this area). Note that a node compromise attack is different from a clone attack. The former usually refers to a case in which the attacker compromises a specific node, and then places that compromised node back into the network, while the latter refers to a case in which the attacker compromises a specific node and places multiple replicated copies (clones) of the compromised node back into the network. Clone attack detection solutions are also different and independent from detecting a single node compromise. This is because of the fact that clone detection methods are usually based on the relations of clone nodes with the same ID with their neighboring nodes, or their placements in the network, and these methods are not capable of detecting "one single" node compromise attack.

Rather than a generic clone model consisting of s *clone groups*, each of which contain at most z clones, we consider a simplified clone model, similar to [11]. In our model, there is only one clone group, with exactly two clones having the same ID. The *clone ID* refers to the ID of two clones in a specific clone group, unless stated otherwise. The use of such a simplified model is to ease the presentation of our main idea, while our method can naturally be applied to a generic clone model without compromising the security.

IV. Preliminaries

Before introducing our proposed clone detection method, we provide a brief background regarding MDS in Section IV-A, which serves as the foundation of our approach. A localization method using MDS that we describe in Section IV-B, called MDS-MAP, provides a core subroutine in our scheme.

A. Multidimensional Scaling

Multidimensional scaling (MDS) [14] is a hyperspace embedding technique, through which pairwise distances are fit into a set of coordinates with the preservation of distance restrictions. More concretely, MDS takes a distance matrix D as input, which is formed from the distances between all pairs of nodes. The output of MDS is a set of coordinates created using only D. The first step is to calculate an inner product matrix $B = CAC$, which satisfies the relation $B = CAC = XX^T$, where $C = I - \frac{1}{n}EE^T$, $A = -\frac{1}{2}D^2$, I is an identity matrix, E is a column vector composed of 1's, and X is a coordinate matrix with each row being a p-dimensional coordinate. One can easily observe that B is a real-valued and symmetric matrix, and hence we can apply orthogonal diagonalization to B to obtain

$$B = QMQ^T, \tag{1}$$

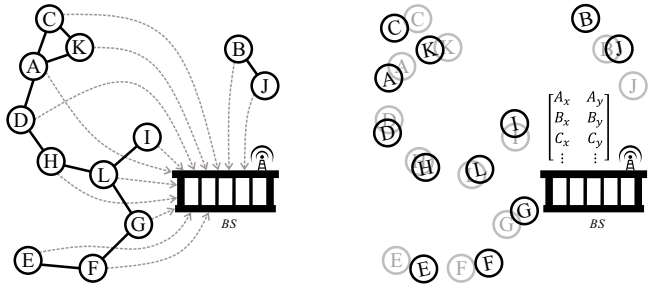
where $M = diag\,(\mu_1,\ \ldots,\ \mu_n)$, each $\mu_i (i \in \{1, ..., n\})$ is an eigenvalue of B, and $Q = [q_1 \cdots q_n]$ is composed of the corresponding orthogonal eigenvectors. Owing to the fact that $B = XX^T$, we can obtain the reconstructed coordinate matrix X' by calculating

$$X' = [q_1 \quad \cdots \quad q_p] \begin{bmatrix} \sqrt{\mu_1} & & \\ & \ddots & \\ & & \sqrt{\mu_p} \end{bmatrix}. \tag{2}$$

However, the coordinate matrix X' reconstructed by MDS is not necessarily identical to X. In essence, X' is only guaranteed to preserve the pairwise distances D, but is subject to translations (shifts), rotations, and reflections. In other words, X and X', where we write $X \neq X'$, can both act as the reconstructed coordinate matrix if X and X' can induce the same D.

B. Localization via MDS

Given a network, MDS-MAP [15] is a localization algorithm executed by the BS. In particular, MDS-MAP takes a subset of pairwise distances of the nodes as input, and generates the coordinates of the nodes in the network. The difference between the ordinary MDS and MDS-MAP lies in the fact that the calculation of MDS assumes that the BS has the knowledge of all pairwise distances. However, this assumption is not realistic, particularly in wireless networks. Thus, MDS-MAP combines the techniques of MDS and a shortest path calculation from graph theory to approximate the ordinary MDS. More specifically, in the case where nodes i and j are far away from each other and i cannot obtain a measured distance from j, the BS instead obtains an approximate $d_{i,j}$ (i.e., the distance between i and j) by calculating the corresponding shortest path. Using this approach, the BS can easily obtain all of the pairwise distances, although some of these are approximate. Next, the BS performs ordinary MDS on the pairwise distances to derive the coordinate matrix and accomplish the localization. Although the approximate distances comprising the input to MDS may cause a certain distortion in the reconstructed coordinates, Shang et al. [15] demonstrated the acceptable reconstruction accuracy of MDS-MAP. Figure 1 shows an illustrative example of the MDS-MAP process. In Figure 1a, each node measures its distance from its neighbors and reports this to the BS. Then, the BS uses the MDS-MAP reconstructed coordinates as the nodes' positions to construct the network map, as shown in Figure 1b.



(a) Nodes send their pairwise distances to the BS.

(b) BS uses MDS's output to rebuild a map.

Fig. 1: An example of the MDS-MAP procedure.

V. Proposed Method

In this section, we describe MDSClone, our proposed method for clone detection. In particular, we explain the basic construction of MDSClone in detail in Section V-A. Then, in Section V-B we describe several improvements to our main construction to yield a more efficient clone detection algorithm. Note that although we mainly use MDS-MAP [15] to calculate the coordinates of IoT devices throughout the paper, we only use the term "MDS" in the remainder of the paper, for representational simplicity.

A. Main Construction of MDSClone

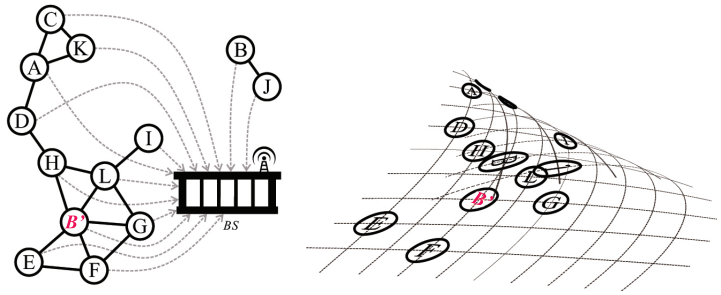
The idea behind our proposed MDS-based solution, MDSClone, is inspired by the following observation: When each node reports its *neighbor-distance information*, consisting of its neighbor list along with the measured pairwise distances, to the BS, the BS can construct a *node map*[8] via MDS without the need to know the exact location information

[8]The terms "node map" and "coordinate matrix" are used interchangeably throughout this paper. While MDSClone detects clones via space distortion, "node map" puts more emphasis on the shape of the reconstructed network. At each time interval, the *ground truth node map* X means the real positions of nodes, while the *reconstructed node map* X' means the estimated positions.

of the nodes. Note that the *node map* refers here to a set of coordinates of IoT devices, and corresponds to the coordinate matrix X in Section IV-A. In the case that no clones are present in the network, a coordinate matrix X' will be generated such that the collected pairwise distances can be approximately preserved (i.e., each entry of the matrix $D - D'$ is close to 0, where D' is the distance matrix calculated from X'). On the other hand, consider a network with a clone. From the BS's perspective, if the information reported from devices collectively contains two nodes with the same ID but completely different neighbor lists, then the reconstructed node map X' will be distorted. More precisely, because two clones can be thought of as two identical nodes that simultaneously appear at two distant locations (e.g., node B in Figure 2a), at least one additional dimension is required in X' to achieve distance preservation. Because the number p of dimensionalities should be a fixed and public parameter, or we may even restrict ourselves to two-dimensional MDS reconstruction ($p = 2$), it follows that a distortion in the reconstructed map is unavoidable (see Figure 2b). As a consequence, from the perspective of clone detection, the failure of MDS in constructing a node map that achieves distance preservation indicates the existence of clones in the network. To identify clones, the BS may execute MDS multiple times, excluding different node IDs. For example, if the MDS calculation for nodes $\{1, \ldots, n\}$ results in an erroneous node map, and the MDS calculation excluding node i (i.e., on nodes $\{1, \ldots, (i-1), (i+1), \ldots, n\}$) achieves a perfect node map reconstruction, then node i must be a clone, because it caused the distortion in the MDS.

In what follows, we describe *three* main design challenges that appear in adopting MDS for clone detection, and explain how MDSClone addresses these challenges.

- First, BS requires the pairwise distances of all the nodes in the network in order to run the MDS algorithm. However, such information is not available. Therefore, the first challenge is to enable the BS to perform the MDS calculation using only a "subset" of pairwise distances. The reason behind this challenge is that in an IoT network, each IoT device can only estimate its distance from its neighboring nodes, e.g., via RSS. Hence, the neighboring information reported to the BS does not include the pairwise distances of all nodes in the network. We address this challenge by using the shortest path between two nodes in order to approximately calculate the Euclidean distance between them (inspired by MDS-MAP [15]).
- The second challenge is to design a localization function (i.e., $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$) in order to "locate" the clones in the network. The reason behind this challenge is that the node map reconstructed by the BS is not necessarily identical to the real positions of nodes (although the pairwise distances are guaranteed to be preserved). In order to address this challenge, we consider two different cases (i.e., the existence of anchor nodes and lack of anchor nodes), as we explain in Section V-A1c.
- The third challenge is the computational overhead imposed on the BS. The reason behind this challenge is that the BS must perform the MDS calculations iteratively in order to find clones. In particular, the BS has to perform, on average, $O(n^c)$ rounds of MDS calculations (where n is the number of nodes in the network and c is the number of clones). We address this challenge by proposing two strategies in MDSClone (as explained in Section V-B): (i) reducing the MDS computational overhead, and (ii) performing the MDS calculation on several server-side devices in a *parallel* manner.



(a) Two nodes with the same ID (nodes B and *B'*).

(b) Distorted reconstructed node map.

Fig. 2: An example IoT network with node B as a clone (we named the clone nodes as B and *B'* for clarification).

1) Detailed Description of MDSClone

The algorithmic description of MDSClone is presented in Algorithm 1. As can be seen, BS is in charge of running the algorithm and recognizing the existence of a clone in the network. Each node i in the network discovers its neighboring nodes N_i, measures the distance $\{(d_{i,j})\}_{j \in N_i}$ with each of its neighboring nodes, and sends this neighbor-distance information $\langle t, i, \{(j, d_{i,j})\}_{j \in N_i} \rangle$ to the BS (comprising the input of Algorithm 1) at time t[9]. Here, the message $\langle t, i, \{(j, d_{i,j})\}_{j \in N_i} \rangle$ can be thought as a star-shaped subgraph whose nodes are $N_i \cup \{i\}$. The purpose of this step is for the BS to collect the subset of pairwise distances, similar to the case in MDS. With the neighbor-distance information of the nodes and a pre-defined distortion threshold (which we will explain in Section V-A1a) as input, the BS periodically executes Algorithm 1.

We assume that the BS maintains a table $\mathcal{L}$ for storing the received neighbor-distance information. After receiving the messages $\{\langle t, i, \{(j, d_{i,j})\}_{j \in N_i} \rangle\}_{i=1,\ldots,n}$, the BS stores the star-shape subgraph induced from $\langle t, i, \{(j, d_{i,j})\}_{j \in N_i} \rangle$ at time t in the t-th row, $\mathcal{L}_t$, of the table $\mathcal{L}$ (steps 2 and 3 of Algorithm 1). Hence, the t-th entry $\mathcal{L}_t$ of the table $\mathcal{L}$ consists of the neighbor-distance information sent by all the nodes at time t. Then, the BS has to perform MDS over $\mathcal{L}_t$ to check whether there are clones present (step 4 of Algorithm 1). More specifically, if the Boolean distortion function $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \tilde{\mathcal{L}}_t)$, that measures the dissimilarity between pairwise distances from $\mathcal{L}_t$ and X'_t (we will explain this function in Section V-A1b) returns true (step 5 of the Algorithm 1), this indicates a significant distortion in the reconstructed node map. In this case, the BS recognizes clones by applying MDS iteratively to $\mathcal{L}_t \setminus \Big((\pi_1, \{(j, d_{\pi_1,j})\}_{j \in N_{\pi_1}}) \ldots (\pi_\rho, \{(j, d_{\pi_\rho,j})\}_{j \in N_{\pi_\rho}}) \Big)$ (steps 6∼8 of Algorithm 1). In essence, this is equivalent to applying MDS over the induced subgraph. During the MDS calculations, after the BS has determined a reconstructed node map whose pairwise distances are not consistent with the collected pairwise distances (step 9 of Algorithm 1), the

[9]Throughout this paper, the notation for time t is sometimes omitted (e.g., the set N_i^t of neighboring nodes and the distance $d_{i,j}^t$ between i and j at time t are abbreviated as N_i and $d_{i,j}$, respectively) without ambiguity, when this can be easily inferred from the context.

Algorithm 1 MDSClone performed by the BS.

1: **Input:** $\langle t, i, \{(j, d_{i,j})\}_{j\in N_i}\rangle$: neighbor-distance information received from node i; λ: distortion threshold.
2: **If** receiving $\{\langle t, i, \{(j, d_{i,j})\}_{j\in N_i}\rangle\}_{i=1,\dots,n}$
3: Update $\mathcal{L}_t$ by calculating $\mathcal{L}_t = (i, \{(j, d_{i,j})\}_{j\in N_i})$
4: $X'_t = \text{MDS}(\mathcal{L}_t)$
5: **If** $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \emptyset) = \texttt{True}$
6: **For** $\rho = 1 \dots n$
7: **For** distinct tuples $\left\{\left((\pi_1, \{(j, d_{\pi_1,j})\}_{j\in N_{\pi_1}}) \dots (\pi_\rho, \{(j, d_{\pi_\rho,j})\}_{j\in N_{\pi_\rho}})\right)\right\}_{\pi_1,\dots,\pi_\rho\in\{1,\dots,n\}}$
8: $X'_t = \text{MDS}\left(\mathcal{L}_t \setminus \left\{\left((\pi_1, \{(j, d_{\pi_1,j})\}_{j\in N_{\pi_1}}) \dots (\pi_\rho, \{(j, d_{\pi_\rho,j})\}_{j\in N_{\pi_\rho}})\right)\right\}_{\pi_1,\dots,\pi_\rho\in\{1,\dots,n\}}\right)$
9: **If** $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \left\{\left((\pi_1, \{(j, d_{\pi_1,j})\}_{j\in N_{\pi_1}}) \dots (\pi_\rho, \{(j, d_{\pi_\rho,j})\}_{j\in N_{\pi_\rho}})\right)\right\}_{\pi_1,\dots,\pi_\rho\in\{1,\dots,n\}}) = \texttt{False}$
10: Nodes $\pi_1, \dots, \pi_\rho$ are identified as clones
11: Calculate $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$ to locate clones $\pi_1, \dots, \pi_\rho$

BS notices that the excluded nodes in the current calculation are clones (step 10 of Algorithm 1). By calculating the localization function $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$, the BS identifies and locates the clones (step 11 of Algorithm 1).

a) Choice of λ: The distortion threshold, λ, controls the trade-off between the tolerance of the MDS reconstruction error, the computational burden on the BS, and the capability of MDSClone to identify clones. In essence, in the case of a small λ value, MDS reconstruction inaccuracies may be regarded as clones, which leads to repeated MDS calculations in order to find these clones. On the other hand, an inappropriately large λ value may result in the misdetection of clones. As can be seen, determining a suitable λ value is important, and in fact a high sensitivity of the MDS reconstruction accuracy leads to a greater capability in identifying clones. In order to address these challenges, in the following we propose a data-driven method to determine an appropriate λ value. Hence, there will be rare cases in which clones could successfully evade detection owing to an inappropriate selection of λ.

Owing to the fact that IoT devices are all owned by the IoT network owner, all the characteristics of the IoT devices, such as their radio ranges, are available to the network owner. Therefore, the network owner is able to virtually deploy IoT devices in a random manner, and then perform the MDSClone calculation on the virtual deployment. Here, we make two observations. First, even in the virtual deployment without clones, the distance matrix D derived directly from $\mathcal{L}_t$ will be slightly different from the distance matrix D' derived from the MDS-MAP reconstruction result. Second, in the presence of clones in the network, the discrepancy between D and D' must be significant, because clones are usually not close to each other, in order to have a more negative impact on the network. As a result, to determine a suitable λ, the network owner sets up x virtual deployments without clones, and chooses the maximum discrepancy value among the x discrepancy values as λ, where x is a sufficiently large value. More precisely, let $\mathcal{A}(D', D)_i$ be given by $\mathcal{A}(D', D) = \frac{\sum_{i<j}(d'_{i,j}-d_{i,j})^2}{1+\dots+(n-1)}$, derived in the i-th virtual deployment (a more detailed description of $\mathcal{A}(D', D)$ can be found in Section V-A1b). The distortion threshold λ is set as the maximum distortion[10], i.e., $\lambda = \max\{\mathcal{A}(D', D)_1, \dots, \mathcal{A}(D', D)_x\}$.

[10]In the execution of step 9 of Algorithm 1, D and D' are not necessarily of dimensions $n \times n$. Thus, one can calculate different λ's for the calculation of $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \bar{\mathcal{L}}_t)$ on D and D' of different sizes. This could constitute a natural extension of the idea behind the calculation of λ here. For the sake of notational simplicity, we omit the detailed explanation of such a fine-grained control of λ in this paper.

b) Construction of $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \bar{\mathcal{L}}_t)$: Similar to λ, the distortion function, i.e., $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \bar{\mathcal{L}}_t)$, has direct impact on the trade-off between the tolerance of MDS reconstruction error, computation burden on the BS, and capability of MDSClone in identifying the clones, but the corresponding construction is still unclear.

Here, we propose an algorithm as an implementation of $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \bar{\mathcal{L}}_t)$. More specifically, the algorithm takes as input a distortion threshold λ, the received neighbor-distance information $\mathcal{L}_t$, and the reconstructed node map X'_t, and outputs an indication of whether the pairwise distances from $\mathcal{L}_t$ are inconsistent with the ones from X'_t. In particular, with the shortest path as an approximation, the BS can calculate pairwise distances $D = \{d_{i,j}\}_{i,j\in\{1,\dots,n\}}$ from $\mathcal{L}_t \setminus \bar{\mathcal{L}}_t$. X'_t can also be used to generate estimated distance matrix $D' = \{d'_{i,j}\}_{i,j\in\{1,\dots,n\}}$. Hence, $\mathcal{D}(\lambda, \mathcal{L}_t, X'_t, \bar{\mathcal{L}}_t)$ returns true (significant inconsistency between D and D', indicating clone existence) if $\mathcal{A}(D', D) = \frac{\sum_{i<j}(d'_{i,j}-d_{i,j})^2}{1+\dots+(n-1)} \geq \lambda$ and false otherwise. Note that $\mathcal{A}(D', D)$ here is defined based on the least square error criterion, while, it is possible to adopt other error metrics.

c) Construction of $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$: After identifying the clones, one option is to announce and revoke clone IDs. In this case, since the nodes physically remain in the network, the attacker might use them in order to conduct other clone attacks or other types of attack, such as blackhole or jamming attacks. Another option is to locate clones and then physically remove them or use one of the existing attestation techniques in the literature (such as [35]). In order to protect the network against further attacks, the latter case is preferable. In this latter case, we need to develop techniques for locating clones in the network, and for that matter we introduce the localization function $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$, which we detail in the following.

We consider two constructions for $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$: (i) Anchor case: a network having at least two static nodes, (ii) No anchor case: without having such restriction. The idea behind the two constructions of $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$ is node alignment: once the real location of nodes at the previous time are known and the alignment between $\mathcal{L}_t$ and $\mathcal{L}_{t-1}$ can be made, we can easily infer the real locations at the current time. In what follows, we explain the considered two scenarios.

Anchor Case. Here, we assume at least two static nodes in the network and we observe that the static nodes can act as anchor points for calculating the transformation matrix P. When P is known, one can obtain the real location of nodes by applying node alignment to $\mathcal{L}_t$ and $\mathcal{L}_{t-1}$.

Consider the case where the BS applies MDS to nodes $1, \ldots, n$. The transformation matrix can also be applied only to a subset $1, \ldots, s$, where $1, \ldots, s$ are static nodes, without loss of generality. In other words, if $X = [X_1^T X_2^T]^T$ and $X' = [X_1'^T X_2'^T]^T$, then $X_1'^T = X_1 P$ and $X_2'^T = X_2 P$. A closer look to the currently used MDS reveals that the MDS works in a restricted form; the mean of reconstructed coordinates of the MDS result will always be shifted to $[0\ 0]$, in the two-dimensional case. This eliminates the need for handling the translation operation, because one can shift the locations at the previous time to be centered at $[0\ 0]$, perform the node alignment, and perform the reverse shift on aligned locations. In this sense, we can therefore take care of the rotation and reflection operations only. Since the MDS is primarily featured by an orthogonal diagonalization (see Section IV-A), we simply find an orthogonal matrix to solve the problem. In particular, we look for an orthogonal matrix P such that $X^{'} = XP$ with the property of

$$X^{'} X^{'T} = XP(XP)^T = XPP^T X^T = XX^T = B. \quad (3)$$

To find such a P, we randomly choose two static nodes from $1, \ldots, s$, and their corresponding reconstructed coordinates. The transformation matrix P can be calculated by considering the relation between two chosen nodes, $\begin{bmatrix} x_1^{'} & y_1^{'} \\ x_2 & y_2 \end{bmatrix} = X' = XP = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} P$, and can be derived as follows,

$$P = \frac{1}{x_2 y_1 - x_1 y_2} \begin{bmatrix} x_2^{'} y_1 - x_1^{'} y_2 & y_2^{'} y_1 - y_1^{'} y_2 \\ x_2 x_1 - x_1 x_2 & y_1 x_2 - y_2 x_1 \end{bmatrix}. \quad (4)$$

Note that the determinant of P, $\det(P)$, may determine the operations applied to nodes. Note also that an odd number of reflections implies $\det(P) = 1$, while the other cases (e.g., only rotation or an even number of reflections) imply $\det(P) = -1$. After obtaining P, we perform node alignment to keep track of the clone movement. In particular, the reconstructed coordinates will be transformed via P to be consistent with the coordinates at the previous time.

No Anchor Case. In this scenario, we do not have anchors and instead seek a construction of $\mathcal{M}(\mathcal{L}_t, \mathcal{L}_{t-1})$ for mapping two sets of points with the guarantee of minimal node-wise discrepancy. The algorithm for the node alignment is inspired by the problem of point set registration. The point set registration [36] has been widely used in computer vision community for finding a spatial transformation that aligns two sets of points. In particular, the point set registration merges two data sets into a globally consistent model by mapping a new measurement to a known data set. The Iterative Closest Point (ICP) algorithm [37] is the most straightforward way for minimizing the difference between two clouds of points. However, ICP works only in the case of a rigid registration (or say, rigid transformation), which typically consists of translation and rotation. The point set registration can also be used in our case where the nodes location at the previous time are mapped to the ones at the current time. However, our problem can be thought as a variant of the conventional point set registration. The differences are stated below:

- While the size of two sets of points in the conventional point set registration problem are allowed to be different, in our case they are guaranteed to be the same.
- While one set of points will be deterministically transformed from another set, in our case due to the consideration of node mobility, we may consider the option of either imperfect transformation or noisy transformation.

Moreover, the MDS-reconstructed node map is subject to not only translation and rotation but also reflection operations, as shown in Section IV-A. Thus, inspired by the robust affine transformation consisting of a richer set of transformations including translation, rotation, and reflection, we resort to affine variant of ICP, *affine ICP*, to find the transformation matrix P between $\{\ell_1^{t-1}, \ldots, \ell_n^{t-1}\}$ and $\{\ell_1^t, \ldots, \ell_n^t\}$, where ℓ_i^{t-1} and ℓ_i^t are the coordinate of the i-th node at $(t-1)$-th time and t-th time, respectively, with $X_{t-1} = [\ell_1^{t-1} \cdots \ell_n^{t-1}]^T$ and $X_t = [\ell_1^t \cdots \ell_n^t]^T$ being the coordinate matrices at $(t-1)$-th time and t-th time, respectively.

Since affine transformation is a richer set of transformations compared to the three transformations required in our context, our problem can still be recasted as the affine registration of two two-dimensional point sets. More formally, based on the least-square error criterion, the affine registration between two point sets can be formulated as

$$\min_{P, j \in \{1,\ldots,n\}} \left(\sum_{i=1}^{n} ||P(\ell_i^{t-1}) - \ell_j^t||_2^2 \right). \quad (5)$$

However, the transformation P in Equation (5) is too vague; it needs to be expressed explicitly so as to simplify the objective function in Equation (5). With the fact that the affine transformation P can be decomposed into an invertible matrix Φ together with a translation vector s, Equation (5) can be rewritten as

$$\min_{\Phi, s, j \in \{1,\ldots,n\}} \left(\sum_{i=1}^{n} ||(\Phi \ell_i^{t-1} + s) - \ell_j^t||_2^2 \right). \quad (6)$$

Note that the presence of anchors eliminates the need of handling coordinate shift but the lack of anchors does not. Furthermore, an invertible real-valued matrix Φ can be decomposed via singular value decomposition (SVD) into $\Phi = USV^T$, where U and V are orthogonal matrices and S is a diagonal matrix with singular values. Because V^T is orthogonal matrix, we define the rotation matrix $R = V^T$. After all, the objective function in Equation (6) can be rewritten as

$$\min_{U, S, R, s, j \in \{1,\ldots,n\}} \left(\sum_{i=1}^{n} ||(USR\ell_i^{t-1} + s) - \ell_j^t||_2^2 \right). \quad (7)$$

Recall that an affine transformation is a combination of a series of basic transformations, such as translation, rotation, and reflection. In Equation (7), U and R represent reflection and rotation matrices, respectively, while S is a scale matrix. Though U and R are orthogonal matrices, the collection of nodes does not scale differently in our consideration. Considering the above constraints, the unconstrained optimization in Equation (7) can be transformed to a constrained optimization,

$$\begin{aligned} \min_{U, R, s, j \in \{1,\ldots,n\}} \quad & \left(\sum_{i=1}^{n} ||(UR\ell_i^{t-1} + s) - \ell_j^t||_2^2 \right) \\ \text{s.t.} \quad & U^T U = I,\ det(U) = 1 \\ & R^T R = I,\ det(R) = 1 \end{aligned}. \quad (8)$$

Inspired by robust affine ICP [36], the optimization in Equation (8) can be solved in the iterative algorithm shown in Algorithm 2. Algorithm 2 outputs rotation matrices U_k, R_k, and translation vector s_k. More specifically, given the $(k-1)$-th transformations $(U_{k-1}, R_{k-1}, s_{k-1})$, the step 3 of Algorithm 2 aims to establish the correspondence of point sets:

$$c_k(i) = \underset{j\in\{1,\dots,n\}}{\operatorname{argmin}} \left(||(U_{k-1}R_{k-1}\ell_i^{t-1} + s_{k-1}) - \ell_j^t||_2^2\right), \quad (9)$$

where $i = 1, \dots, n$. Note that, step 3 of Algorithm 2 can be solved by methods such as *k-d tree* [38] and *nearest point search based on Delaunay tessellation* [39]. Given the found correspondence $c_k(i)$, step 5 of Algorithm 2 computes the k-th transformations (U_k, R_k, s_k):

$$(U_k, R_k, s_k) = \underset{U\in\mathfrak{L}(2), R\in\mathfrak{L}(2), s\in\mathbb{R}^2}{\operatorname{argmin}} \left(\sum_{i=1}^{n} ||UR\ell_i^{t-1} + s - \ell_i^t||_2^2\right), \quad (10)$$

where $\mathfrak{L}(2) = \{\Upsilon \in \mathbb{R}^{2\times 2} | \Upsilon^T\Upsilon = I_2, \det(\Upsilon) = 1\}$ denotes an orthogonal group consist of a set of real 2×2 orthogonal matrices. Considering the Taylor series of exponential mappings on $U_k, R_k \in \mathfrak{L}(2)$, step 5 of Algorithm 2 can be formulated and solved using quadratic programming.

Algorithm 2 Node alignment based on affine ICP.

1: **INPUT:** k is initialized as 0; U_0, R_0, and s_0 are randomly chosen; ϵ is the considered threshold.
2: **Repeat**
3: Compute the correspondence (Equation (9)) by using $U_{k-1}, R_{k-1}, s_{k-1}$
4: Compute the error function $\epsilon_k(U, R, s) = \frac{1}{n}\sum_{i=1}^{n} ||(U_{k-1}R_{k-1}\ell_i^t + s_{k-1}) - \ell_j^{t-1}||_2^2$
5: Compute the k-th transformation (U_k, R_k, t_k) and $k = k + 1$
6: **Until** $|\epsilon_k(U_k, R_k, s_k) - \epsilon_{k-1}(U_{k-1}, R_{k-1}, s_{k-1})| \leq \epsilon$ or k reaches a predefined threshold

Lemma 1. *Algorithm 2 converges monotonically to a local minimum, with respect to the mean squared distance from any given initial parameters.*

Algorithm 2 shares the same spirit as robust affine ICP [36]. Nevertheless, due to its similarity to robust affine ICP and the lack of space, we omit the formal proof of Lemma 1 here.

B. Techniques for Efficiency Improvement of MDSClone

As we explained in Section V-A, the BS should execute the Algorithm 1 in order to check if the network is under clone attack, and in case of having clones, BS should execute MDS function (steps 7 to 10 of Algorithm 1) multiple times to identify the clone IDs. In such situation, BS has to perform, on average, $O(n^c)$ rounds of MDS calculations[11] to find the clones, provided that c clones exist in the network. Though the MDS computation is very fast, the iterative calculation of MDS may still impose a huge computation overhead on the BS. For example, the execution of the MDS on a matrix of dimension $10^4 \times 10^4$ takes approximately two minutes. In the case of only one clone in a network of 10^4 IoT devices, the BS needs to perform the MDS, on average, 5×10^3 times, which requires hours or even days for detecting the clones. In what follows, in order to address this issue, we propose an improvement to the calculation of the MDS function. In particular, we show that the MDS calculation can be parallelized and offloaded on several powerful servers, or devices, each of which calculating one of the required iterations that results in speeding up the whole clone detection algorithm. We show that our proposal significantly reduces the computation load on the BS, leading to improved scalability and performance of the clone detection method. We provide detailed explanation of our improved implementation in the following.

[11] In the extreme case, assume that there is one clone group consists of c clones in the network. Since these c clones share the same ID (where ID$\in \{1, \dots, n\}$), the MDSClone has to perform the MDS calculation for $O(n)$ times (on average $\frac{n}{2}$ times) to identify the clone ID. Consider another extreme case, where c clone groups exist in the network. Since the BS has no knowledge on the number of clones (i.e., c), the BS has to scan every possible combination of clone IDs, leading to $O(n)+O(n^2)+\cdots+O(n^c) = O(n^c)$ times of MDS calculations, where $O(n^i)$ comes with $\binom{n}{i}$ possibilities of one clone ID (where $i = 1 \dots c$).

1) Speeding Up the MDS Calculation

In general, a significant portion of computation overhead in the MDS calculation is incurred by computing eigenpairs. Here, the eigenpairs mean the pairs of eigenvalues and eigenvectors in Equation (1). In addition, obviously the computation load in the MDS is proportional to the number of nodes in the network. To ensure the scalability of MDSClone, with the observation that the inner product matrix B in our context is always real-valued and symmetric, we propose three techniques to improve the performance of the MDS calculation in MDSClone algorithm:

a) *CIPMLO:* aims at computing inner product matrix with less arithmetic operations.
b) *TI:* is using modified Householder transformation [40] to speed-up the calculation of eigenpairs.
c) *SMEBM:* is closely relevant to *TI*, and basically speeds-up *TI*.

Each of these three techniques leads to certain extent of the speed-up. Among them, *CIPMLO* and *TI* can be executed individually, while *SMEBM* is useful only when it is used along with *TI*. We detail each of these three techniques in the following.

*a) **Computing Inner Product Matrix With Less Operations (CIPMLO)**:* This technique, computes inner product matrix B with less arithmetic operations. Starting with a concrete example is helpful in giving an idea of how *CIPMLO* achieves the speed-up. Assume that we have a distance matrix D such that

$$D^2 = \begin{bmatrix} 0 & 10 & 34 & 20 \\ 10 & 0 & 52 & 50 \\ 34 & 52 & 0 & 10 \\ 20 & 50 & 10 & 0 \end{bmatrix}. \quad (11)$$

We derive matrices C and A as follows,

$$C = \frac{1}{4}\begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix} \text{ and}$$

$$A = \begin{bmatrix} 0 & -5 & -17 & -10 \\ -5 & 0 & -26 & -25 \\ -17 & -26 & 0 & -5 \\ -10 & -25 & 5 & 0 \end{bmatrix}. \quad (12)$$

We calculate the matrix $\Omega = C \cdot A$. A partial view of Ω is shown in the following,

$$\Omega = \frac{1}{n}\begin{bmatrix} 5+17+10 & \cdots \\ -15+17+10 & \cdots \\ 5-51+10 & \cdots \\ 5+17-30 & \cdots \end{bmatrix}. \quad (13)$$

With the observation that, the elements in the i-th column of Ω are related to the element $\Omega_{i,i}$, we can derive non-diagonal

elements by using diagonal elements. In essence, the matrix Ω can be represented in a more formal way as follows.

$$\Omega = \frac{-1}{n}\begin{bmatrix} D_1 & D_2 - nN_1 & D_3 - nN_2 & D_4 - nN_3 \\ D_1 - nN_1 & D_2 & D_3 - nN_4 & D_4 - nN_5 \\ D_1 - nN_2 & D_2 - nN_4 & D_3 & D_4 - nN_6 \\ D_1 - nN_3 & D_2 - nN_5 & D_3 - nN_6 & D_4, \end{bmatrix}, \tag{14}$$

where $D_i = \sum_{j=1}^{n} A_{j,i}$ and N_k is the non-diagonal element in the lower triangular part of A, with the elements of A being re-numbered as

$$\begin{bmatrix} 0 & & & & \\ N_1 & 0 & & & \\ N_2 & N_n & \ddots & & \\ \vdots & \vdots & \vdots & 0 & \\ N_{n-1} & N_{2n-3} & & N_{\frac{n(n-1)}{2}} & 0 \end{bmatrix}. \tag{15}$$

Let Ω be

$$\Omega = \frac{-1}{n}\begin{bmatrix} e_1 & e_5 & e_9 & e_{13} \\ e_2 & e_6 & e_{10} & e_{14} \\ e_3 & e_7 & e_{11} & e_{15} \\ e_4 & e_8 & e_{12} & e_{16} \end{bmatrix}. \tag{16}$$

Let S be $-(e_1 + e_5 + e_9 + _{13})$. Recall that $B = \Omega C = CAC$ (see Section IV-A), we obtain

$$B = \frac{-1}{n^2}\begin{bmatrix} 4e_1 + S & 4e_5 + S & 4e_9 + S & 4e_{13} + S \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}. \tag{17}$$

Similarly, we find that each element of B has relation with the sum of row elements of Ω. In fact, B can be expressed in the following form,

$$B = \frac{-1}{n^2}\begin{bmatrix} ne_1 - R_1 & ne_5 - R_1 & ne_9 - R_1 & ne_{13} - R_1 \\ ne_2 - R_2 & ne_6 - R_2 & ne_{10} - R_2 & ne_{14} - R_2 \\ ne_3 - R_3 & ne_7 - R_3 & ne_{11} - R_3 & ne_{15} - R_3 \\ ne_4 - R_4 & ne_8 - R_4 & ne_{12} - R_4 & ne_{16} - R_4 \end{bmatrix}, \tag{18}$$

where R_i is defined as $R_i = S - nD_i$ with $S = \sum_{j=1}^{n} D_i$. Then we prove that R_i is the sum of the i-th row of Ω.

Lemma 2. *R_i is the sum of the i-th row of Ω.*

Proof. $R_i = S - nD_i = \sum_{i=1}^{n} D_i - n\sum_{j=1}^{n} A_{j,i}$
$= \sum_{j=1}^{n} \Omega_{i,j}$. □

Then, we prove that ΩC calculated in our proposed procedures remains symmetric.

Lemma 3. *ΩC is symmetric.*

Proof. $\frac{-1}{n}\Omega C_{i,j} = \frac{-1}{n}(n\Omega_{i,j} - R_i) = \frac{-1}{n}[n(D_j - nA_{i,j}) - (S - nD_i)] = \frac{-1}{n}[n(D_i - nA_{j,i}) - (S - nD_j)] = \frac{-1}{n}(n\Omega_{j,i} - R_j) = \frac{-1}{n}\Omega C_{j,i}$, where the first equality and second equality are due to equations (18) and (14), respectively. □

Since $B = \Omega C$ is symmetric, we can calculate the elements of only upper or lower triangular part. In addition, we find that the upper triangular part of B in Equation (18) is only dependent on the upper triangular part of Ω in Equation 16. In other words, we can calculate only a half of elements of Ω and then compute B with approximately half of the computing burden, compared to the original computation task. Such calculation of B is shown below.

$$\begin{aligned} B &= -\frac{1}{n^2}\Omega C \\ &= -\frac{1}{n^2}\begin{bmatrix} D_1 & D_2 - nN_1 & D_3 - nN_2 & D_4 - nN_3 \\ \cdot & D_2 & D_3 - nN_4 & D_4 - nN_5 \\ \cdot & \cdot & D_3 & D_4 - nN_6 \\ \cdot & \cdot & \cdot & D_4 \end{bmatrix} C \\ &= -\frac{1}{n^2}\begin{bmatrix} ne_1 - R_1 & ne_5 - R_1 & ne_9 - R_1 & ne_{13} - R_1 \\ \cdot & ne_6 - R_2 & ne_{10} - R_2 & ne_{14} - R_2 \\ \cdot & \cdot & ne_{11} - R_3 & ne_{15} - R_3 \\ \cdot & \cdot & \cdot & ne_{16} - R_4 \end{bmatrix}. \end{aligned} \tag{19}$$

Note that the third equality is due to Equation (18).

*b) **Tridiagonalization Improvement (TI)**:* Before calculating eigenvalue decomposition, the existing library for eigenvalue computation [41] introduces a pre-processing phase for computation reduction. In particular, when the matrix is symmetric, one can apply *Householder transform* to the input matrix and obtain a tridiagonal matrix. After that, the eigenvalue decomposition applies to derive the eigenvalues and eigenvectors. In our context, we focus only on B, which is naturally symmetric. Consequently, our second proposed technique, *TI*, to speed-up the MDS calculation is due to the performance improvement of matrix tridiagonalization.

Basically, *TI* achieves the speed-up because some matrix multiplications can be replaced by matrix additions and inner product calculations. We also start with how Householder transform works, to better illustrate the basic idea behind the design. Let $A = A_0$ be a real-valued symmetric matrix of dimension $n \times n$. We can reduce A to the tridiagonal form A_{n-2} by iteratively using decomposition, $A_m = P_m A_{m-1} P_m$, $m = 1 \ldots (n-2)$, where $P_m = \begin{bmatrix} H_{n-m} & 0 \\ 0 & I_m \end{bmatrix}$ is an orthogonal matrix with I_m being an $m \times m$ identity matrix and H_{n-m} being a Householder matrix defined as $H_{n-m} = I_{n-m} - 2v_{n-m}v_{n-m}^T$. Here, $v_{n-m} = \frac{x_{n-m} - \sigma e_{n-m}}{\|x_{n-m} - \sigma e_{n-m}\|}$ is a Householder vector, where x_{n-m} is a column vector composed of the first $n - m$ elements of the m-th column of A, $\sigma = -sign(x_\bullet)\|x_{n-m}\|$ is the length of x with $x_\bullet$ being the last element of x_{n-m}, $e_{n-m} = (0,\ 0,\ \ldots,\ 1)^T$ is the last standard basis vector of dimension $(n-m) \times 1$, and $sign(x_\bullet)$ is defined as

$$sign(x_\bullet) = \begin{cases} 1 & \text{if } x_1 \geq 0 \\ -1 & \text{if } x_1 < 0 \end{cases}. \tag{20}$$

Afterwards, we can generate the tridiagonal form of a given symmetric matrix. More concretely, consider the first iteration of Householder transformation of A,

$$A = A_0 = \begin{bmatrix} e_1 & e_2 & e_3 & e_4 \\ e_2 & e_5 & e_6 & e_7 \\ e_3 & e_6 & e_8 & e_9 \\ e_4 & e_7 & e_9 & e_{10} \end{bmatrix}. \tag{21}$$

Let $x_{n-1} = \begin{bmatrix} e_4 \\ e_7 \\ e_9 \end{bmatrix}$ and $v_{n-1} = \frac{\hat{v}_{n-1}}{sqrt}$, where $\hat{v}_{n-1} = \begin{bmatrix} e_4 \\ e_7 \\ (e_9 + \sigma) \end{bmatrix}$ and $sqrt = \sqrt{e_4^2 + e_7^2 + (e_9 + \sigma)^2}$. Then, we obtain

$$H_{n-1} = I_{n-1} - \frac{2}{sqrt^2}\begin{bmatrix} e_4^2 & e_4 e_7 & e_4(e_9 + \sigma) \\ e_7 e_4 & e_7^2 & e_7(e_9 + \sigma) \\ (e_9 + \sigma)e_4 & (e_9 + \sigma)e_7 & (e_9 + \sigma)^2 \end{bmatrix}, \tag{22}$$

and

$$P_1 = \begin{bmatrix} 1 - \frac{2e_4^2}{sqrt^2} & \frac{-2e_4 e_7}{sqrt^2} & \frac{-2e_4(e_9+\sigma)}{sqrt^2} & 0 \\ \frac{-2e_4 e_7}{sqrt^2} & 1 - \frac{2e_7^2}{sqrt^2} & \frac{-2e_7(e_9+\sigma)}{sqrt^2} & 0 \\ \frac{-2e_4(e_9+\sigma)}{sqrt^2} & \frac{-2e_7(e_9+\sigma)}{sqrt^2} & 1 - \frac{2(e_9+\sigma)^2}{sqrt^2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{23}$$

As concrete examples, we list some elements of $P_1 A$.

$$(P_1 A)_{1,1} = \underbrace{\frac{\overbrace{-2}^{(a)}\, e_4 \overbrace{[e_1 e_4 + e_2 e_7 + e_3(e_9 + \sigma)]}^{(c)}}{\underbrace{sqrt^2}_{(a)}}}_{(b)} + e_1$$

$$\text{and } (P_1 A)_{2,1} = \underbrace{\frac{\overbrace{-2}^{(a)}\, e_7 \overbrace{[e_1 e_4 + e_2 e_7 + e_3(e_9 + \sigma)]}^{(c)}}{\underbrace{sqrt^2}_{(a)}}}_{(b)} + e_2, \tag{24}$$

$$(P_1A)_{1,2} = \frac{\overbrace{-2}^{(a)}\, e_4 \overbrace{[e_2e_4 + e_5e_7 + e_6(e_9+\sigma)]}^{(d)}}{\underbrace{sqrt^2}_{(a)}} + \underbrace{e_2}_{(b)}$$

$$\text{and } (P_1A)_{2,2} = \frac{\overbrace{-2}^{(a)}\, e_7 \overbrace{[e_2e_4 + e_5e_7 + e_6(e_9+\sigma])}^{(d)}}{\underbrace{sqrt^2}_{(a)}} + \underbrace{e_5}_{(b)}. \quad (25)$$

Before describing our approach based on modified Householder transformation, we highlight some observations from equations (24) and (25) as follows:

1) The elements with the same column have similar calculations.
2) All elements of P_1A share some common operations (part (a)).
3) The elements in part (b) are the elements from matrix A.
4) The calculations in part (c) are the same, as well as the calculations in part (d).
5) The remaining parts, e_4 and e_7, are elements of $\hat{v}_{n-1}$.

TI mainly speeds-up some parts of the numerators of the elements in P_1A (e.g., parts (c) and (d)), but for the time being, we only focus on part (c). Considering $e_1e_4+e_2e_7+e_3(e_9+\sigma)$, we recognize that this it is the inner product of $\hat{v}_{n-1}^T$ and $\begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}^T$ (i.e., part of matrix A). So, P_1A can be represented as

$$\begin{bmatrix} \frac{-2\hat{v}_{n-1}(1)(X_{123})}{sqrt^2} + e_1 & \frac{-2\hat{v}_{n-1}(1)(X_{256})}{sqrt^2} + e_2 & \cdots \\ \frac{-2\hat{v}_{n-1}(2)(X_{123})}{sqrt^2} + e_2 & \frac{-2\hat{v}_{n-1}(2)(X_{256})}{sqrt^2} + e_5 & \cdots \\ \frac{-2\hat{v}_{n-1}(3)(X_{123})}{sqrt^2} + e_3 & \frac{-2\hat{v}_{n-1}(3)(X_{256})}{sqrt^2} + e_6 & \cdots \\ e_4 & e_7 & \cdots \end{bmatrix}, \quad (26)$$

Where $\hat{v}_{n-1}(i)$ is the i-th element of $\hat{v}_{n-1}$, $X_{123} = \hat{v}_{n-1}^T \cdot \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}^T$, and $X_{256} = \hat{v}_{n-1}^T \cdot \begin{bmatrix} e_2 & e_5 & e_6 \end{bmatrix}^T$. Let V_r be the inner product of $\hat{v}_{n-1}^T$ and $\begin{bmatrix} A_{r,1} & A_{r,2} & A_{r,3} \end{bmatrix}$. Then, P_1A can be rewritten as

$$\frac{-2}{sqrt^2}\left[\begin{bmatrix} \hat{v}_{n-1} \\ 0 \end{bmatrix} \otimes ([V_1E \quad V_2E \quad V_3E \quad V_4E])\right] + A, \quad (27)$$

where $\otimes$ denotes the element-wise multiplication and E is the vector of all 1's. By considering equations (23), (24), and (25), the same observations and procedures can also be applied to P_1AP_1 (= A_1) in order to obtain

$$A_{1_{(1,1)}} = \underbrace{S}_{(e)}\left[\underbrace{S}_{(e)}\underbrace{\hat{v}_{n-1}(1)\bar{V_1}}_{(f)} + \hat{v}_{n-1}(1)V_1 + V_1\hat{v}_{n-1}(1)\right] + \underbrace{e_1}_{(g)}, \quad (28)$$

$$A_{1_{(2,1)}} = \underbrace{S}_{(e)}\left[\underbrace{S}_{(e)}\underbrace{\hat{v}_{n-1}(2)\bar{V_1}}_{(f)} + \hat{v}_{n-1}(2)V_1 + V_2\hat{v}_{n-1}(1)\right] + \underbrace{e_2}_{(g)}, \quad (29)$$

where $S = \frac{-2}{sqrt^2}$ and $\bar{V_1} = \hat{v}_{n-1}(1)\hat{v}_{n-1}^T \cdot \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix}^T$. We can see from equations (28) and (29) that for the elements of the same column in A_1, they perform the same operations. For example, the multiplication with S (part (e)) and an inner product (part (f)) are common operations. We can also see that the elements in part (g) are from matrix A. Let $\bar{V_i}$ be $\hat{v}_{n-1}(i)\hat{v}_{n-1}^T \cdot \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix}^T$. Thus, A_1 can be formulated as

$$\begin{aligned} A_1 =& S^2\left[\begin{bmatrix} \hat{v}_{n-1} \\ 0 \end{bmatrix} \otimes [\bar{V_1}E \quad \bar{V_2}E \quad \bar{V_3}E \quad 0E]\right] + \\ & S\left[\begin{bmatrix} \hat{v} \\ 0 \end{bmatrix}\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^T + \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}\begin{bmatrix} \hat{v} \\ 0 \end{bmatrix}^T\right] + A. \end{aligned} \quad (30)$$

Then, in the following lemma, we prove that A_1 calculated in the above way is symmetric.

Lemma 4. *A_1 calculated in the above way is symmetric.*

Proof. Let $w = [\hat{v}_{n-1} \quad 0]^T$ and $W = [\bar{V_1} \quad \bar{V_2} \quad \bar{V_3} \quad 0]^T$. $A_{1_{(i,j)}} = S^2[w(i)W(j)] + S[w(i)V(j) + V(i)w(j)] + A_{i,j} = S^2[w(i)W(j)] + [w(i)V(j) + V(i)w(j)] = S^2[\hat{v}_{n-1}(i)\hat{v}_{n-1}(j)(\hat{v}_{n-1}^T \cdot [V_1 \quad V_2 \quad V_3]^T)] = \hat{v}_{n-1}(i)\hat{v}_{n-1}(j)\hat{v}_{n-1}^T \cdot [V_1 \quad V_2 \quad V_3]^T) = S^2[\hat{v}_{n-1}(j)\hat{v}_{n-1}(i)(\hat{v}_{n-1}^T \cdot [V_1 \quad V_2 \quad V_3]^T)] = S^2[w(j)W(i)] + [V(i)w(j) + w(i)V(j)] = S^2[w(j)W(i)] + S[w(j)V(i) + V(j)w(i)] + A_{j,i} = A_{1_{(j,i)}}$. □

According to the property of Householder transformation, after the first iteration, A_1 can be expressed in the following form,

$$\begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & e \\ 0 & 0 & e & d \end{bmatrix}. \quad (31)$$

In the following lemma, we formally prove that the upper-right entry of A_1 calculated in the above way is zero.

Lemma 5. *The upper-right entry of A_1 calculated in the above way is zero.*

Proof. Assume $(P_1AP_1)_{i,j} = 0$, where $i \leq j-2$ and $j \geq n-2$. $A_{1_{(i,j)}} = S[\hat{v}_{n-1}(i)V_j] + A_{i,j} = S\left[\hat{v}_{n-1}(i)\left(\hat{v}_{n-1}^T \cdot [A_{j,1} \quad A_{j,2} \quad A_{j,3}]^T\right)\right] + A_{i,j} = S\left[\hat{v}_{n-1}(i)\left(\hat{v}_{n-1}(1)A_{j,1} + \hat{v}_{n-1}(2)A_{j,2} + \hat{v}_{n-1}(3)A_{j,3}\right)\right] + A_{i,j} = S\left[A_{i,j}\left(A_{1,j}A_{j,1} + A_{2,j}A_{j,2} + (A_{3,j}+\sigma)A_{j,3}\right)\right] + A_{i,j} = S\left[A_{i,j}\left(A_{1,j}^2 + A_{2,j}^2 + A_{3,j}^2 + A_{j,3}\sigma\right)\right] + A_{i,j} = S\left[A_{i,j}\left(\sigma^2 + A_{j,3}\sigma\right)\right] + A_{i,j} = S\left[A_{i,j}(-S^{-1})\right] + A_{i,j} = -A_{i,j} + A_{i,j} = 0$ □

We can see from Equation (30), Lemma 4, and Lemma 5 that the original matrix multiplications in Householder transform can be replaced by matrix additions and inner product calculations. The same observations and procedures can also be applied to A_2, A_3, and so on. As a consequence, we have the following corollary.

Corollary 1. *The matrix multiplications involved in the calculation of A_i, where $i = 1 \ldots (n-2)$, can be replaced by inner product and matrix addition calculations.*

Here, our proposed *TI* for Householder transformation achieves the speed-up by taking advantage of the computationally cheaper inner product and matrix addition calculations. Compared to the current software implementation, EISPACK [41], our proposed algorithm needs only half computation.

*c) **Searching for Meaningful Eigenpairs of Block Matrix (SMEBM)**:* If the reconstructed coordinates are constrained to be two-dimensional, in fact, two largest eigenpairs suffice to reconstruct X'. Note that "meaningful eigenpairs" here refer to those eigenpairs that have impact on the coordinates of our interest. More specifically, as stated previously, we perform tridiagonalization and eigenvalue decomposition to derive X'. However, we observe that the two-dimensional node map is only affected by specific elements. As a result, the idea behind *SMEBM* is that we only calculate the entries that may affect the eigenpairs that have impact on the two-dimensional coordinates. One distinguishing feature of *SMEBM* is that one may need $n-2$ iterations of calculations of A_1, ..., A_{n-2} in *TI*, however, only two or three iterations in *SMEBM* are needed.

Given that all the elements of $A = A_0$ are very small, we observe from the tridiagonalized matrix that only some values of small block submatrix are meaningful. In particular, there are two forms of block submatrices.

$$\begin{bmatrix} D_i & E_{i-1} & 0 & \cdots & 0 \\ E_{i-1} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & d_3 & e_2 & 0 \\ \vdots & \ddots & e_2 & d_2 & e_1 \\ 0 & \cdots & 0 & e_1 & d_1 \end{bmatrix} \text{ and } \begin{bmatrix} d_i & e_{i-1} & 0 & \cdots & 0 \\ e_{i-1} & d_{i-1} & 0 & \ddots & \vdots \\ 0 & 0 & \ddots & 0 & 0 \\ \vdots & \ddots & 0 & d_2 & e_1 \\ 0 & \cdots & 0 & e_1 & d_1 \end{bmatrix}. \tag{32}$$

The first form is a 3×3 block submatrix (see the former part of Equation (32)), containing two meaningful eigenvalues, where an eigenvalue is close to zero and will always be at the bottom-right corner. The second form is composed of two 2×2 block submatrices (see the latter part of Equation (32)): they can be found in diagonal part of tridiagonalized matrix, an one of them is guaranteed to be at the bottom-right corner. Furthermore, in the case of two 2×2 block submatrices, each block submatrix contributes one meaningful eigenpairs. The reason why we can only focus on the block submatrix is that all the elements of tridiagonal matrix are close to zero except the elements of block submatrices.

To better illustrate the idea behind *SMEBM*, we start with a concrete example. Consider a 5×5 tridiagonal matrix in Equation (33) with a 3×3 block submatrix and "$*$" elements between 10^{-10} and 10^{-16}.

$$\begin{bmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & d_3 & e_2 & 0 \\ 0 & 0 & e_2 & d_2 & e_1 \\ 0 & 0 & 0 & e_1 & d_1 \end{bmatrix}. \tag{33}$$

Since "$*$" is very close to 10^{-10} and 10^{-16}, we consider them to be zero. So, the eigenvalues of tridiagonal matrix Δ can be easily extended to be the eigenvalues of matrix $\begin{bmatrix} O & 0 \\ 0 & \Delta \end{bmatrix}$ by padding zero submatrix O. The above works only in the case where each element is sufficiently small. Nevertheless, a preprocessing step can be used to counteract the above problem. After calculating the inner product matrix, we scale it by calculating $\frac{B}{\alpha}$, where $\alpha = \sum_{d=1}^{n} B_{d,d} + \sum_{r=1}^{n-1} \sum_{c=r+1}^{n} B_{r,c}$. Because of the above preprocessing on the inner product matrix B, the eigenvalues calculated after the scaling, the values of μ_i', will not be the same as original ones, μ_i. However, μ_i can be recovered by calculating $\mu_i = \alpha\mu_i'$, $i = 1 \ldots n$. Since we only need the block submatrix, we can reduce some steps when performing Householder transformation. In particular, for a 3×3 block submatrix, we just need to transform the inner product matrix twice. For two 2×2 block submatrices, we need to transform the inner product matrix two or three times.

$$\begin{bmatrix} * & * & * & * & + \\ * & * & * & * & + \\ * & * & * & * & + \\ * & * & * & * & + \\ + & + & + & + & d_1 \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & * & + & 0 \\ * & * & * & + & 0 \\ * & * & * & + & 0 \\ + & + & + & d_2 & e_1 \\ 0 & 0 & 0 & e_1 & d_1 \end{bmatrix}$$

matrix A — 1st iteration (A_1)

$$\Rightarrow \begin{bmatrix} ? & ? & ? & 0 & 0 \\ ? & ? & ? & 0 & 0 \\ ? & ? & d_3 & e_2 & 0 \\ 0 & 0 & e_2 & d_2 & e_1 \\ 0 & 0 & 0 & e_1 & d_1 \end{bmatrix} \tag{34}$$

2nd iteration (A_2)

From Equation (34), we can see that, after one iteration of Householder transformation, the positions with "$+$" mark will become zero, and the elements on the positions with "$*$" mark will be changed because they are multiplied by Householder matrix. In the case of a 3×3 block submatrix, e_2 must be greater than $\varepsilon \approx 10^{-10}$ and we do not need to compute the remaining parts with "?" mark. In the case of $e_2 \leq \varepsilon$, we deflate the matrix, resulting in the case of two 2×2 block submatrices. Basically, we check whether the diagonal elements are greater than ε. To search for the two 2×2 block submatrices, we have three different cases as follows.

1) One 2×2 block submatrix is at the bottom-right corner, while another is 1×1 block submatrix (or say, 2×2 degenerated block submatrix with the bottom-right element being 0) at the upper-left corner.
2) One 2×2 block submatrix is at the bottom-right corner, while another is 2×2 block submatrix at the upper-left corner.
3) One 2×2 block submatrix is at the bottom-right corner, while another is 2×2 block submatrix at the matrix diagonal part (but not at the upper-left corner).

elements with "$\diamond$" mark are ≈ 0

$$\begin{bmatrix} d_i & \diamond & \diamond & 0 & 0 \\ \diamond & \leq\varepsilon & \diamond & 0 & 0 \\ \diamond & \diamond & \leq\varepsilon & \leq\varepsilon & 0 \\ 0 & 0 & \leq\varepsilon & d_2 & e_1 \\ 0 & 0 & 0 & e_1 & d_1 \end{bmatrix} \begin{bmatrix} d_i & e_{i-1} & \diamond & 0 & 0 \\ e_{i-1} & d_{i-1} \geq\varepsilon & \diamond & 0 & 0 \\ \diamond & \diamond & \leq\varepsilon & \leq\varepsilon & 0 \\ 0 & 0 & \leq\varepsilon & d_2 & e_1 \\ 0 & 0 & 0 & e_1 & d_1 \end{bmatrix}$$

first case (A_2) — second case (A_2)

square matrix

$$\begin{bmatrix} ? & ? & ? & 0 & 0 \\ ? & ? & ? & 0 & 0 \\ ? & ? & >\varepsilon & \leq\varepsilon & 0 \\ 0 & 0 & \leq\varepsilon & d_2 & e_1 \\ 0 & 0 & 0 & e_1 & d_1 \end{bmatrix} \tag{35}$$

third case (A_2)

The above algorithm for 5×5 matrix can be extended to handle $n \times n$ matrix. Algorithm 3 searches for meaningful eigenpairs with the minimal number of iterations of Householder transformation. Two largest eigenpairs among three calculated eigenpairs in the case of a 3×3 meaningful block submatrix or among four calculated eigenpairs in the case of two 2×2 meaningful block submatrices, in fact, suffice to reconstruct X'.

VI. Experimental Analysis

In order to evaluate the performance of MDSClone in detecting clones, we have conducted several experimental analyses considering various network settings and evaluation criteria (i.e., detection probability, computation time, and memory and energy consumption). To study the practicality of our proposed MDSClone scheme for the current generation of sensors in an IoT environment, we have implemented

Algorithm 3 Calculating meaningful eigenpairs.

SETTING: $(A_i)_{x,y}$ is the element of A_i at the intersection of x-th row and y-th column
SETTING: $A_{[(i,j),(x,y)]}$ is submatrix whose upper-left (bottom-right) element is $A_{i,j}$ ($A_{x,y}$)
INPUT: $A \in \mathbb{R}^{n \times n}$ with small values as matrix elements
  Calculate $A_1 = \text{HOUSEHOLDER}(A)$ and $A_2 = \text{HOUSEHOLDER}(A_1)$
  if $(A_2)_{n-2,n-1} \geq \varepsilon$
    $(A_2)_{[(n-2,n-2),(n,n)]}$ is the 3×3 block submatrix
  else
    for $m = 2$ to $n - 3$
      if $(A_2)_{n-m,n-m} \geq \varepsilon$
        $X = \text{HOUSEHOLDER}((A_2)_{[(1,1),(n-m,n-m)]})$
        $(X)_{[(n-m-1,n-m-1),(n-m,n-m)]}$ and $(A_2)_{[(n-1,n-1),(n,n)]}$ are two 2×2 block submatrices
      if $(A_2)_{2,2} \geq \varepsilon$
        $(A_2)_{[(1,1),(2,2)]}$ and $(A_2)_{[(n-1,n-1),(n,n)]}$ are two 2×2 block submatrices
      if $(A_2)_{1,1} \geq \varepsilon$
        $(A_2)_{[(1,1),(1,1)]}$ is a 1×1 submatrix and $(A_2)_{[(n-1,n-1),(n,n)]}$ is a 2×2 block submatrix

a prototype of our scheme on TelosB motes running TinyOS (with the following specifications: Micro-Controller: TI MSP430F1611; ROM: 48KB+256B; RAM: 10KB; Radio Chipset: ChipCon CC2420). We executed our algorithm using TOSSIM [42] on TinyOS 1.1.15 to evaluate the energy consumption of MDSClone. Note that TOSSIM is a discrete-event simulator, designed especially for TinyOS WSNs, on which TinyOS code can be executed directly. Owing to this feature, although TOSSIM is in essence a simulator, its estimation of energy consumption is rather accurate. In our experimental setting, we considered different network sizes varying from 1,000 to 10,000. Moreover, we considered different numbers of clone groups in the network, varying from two to 14.

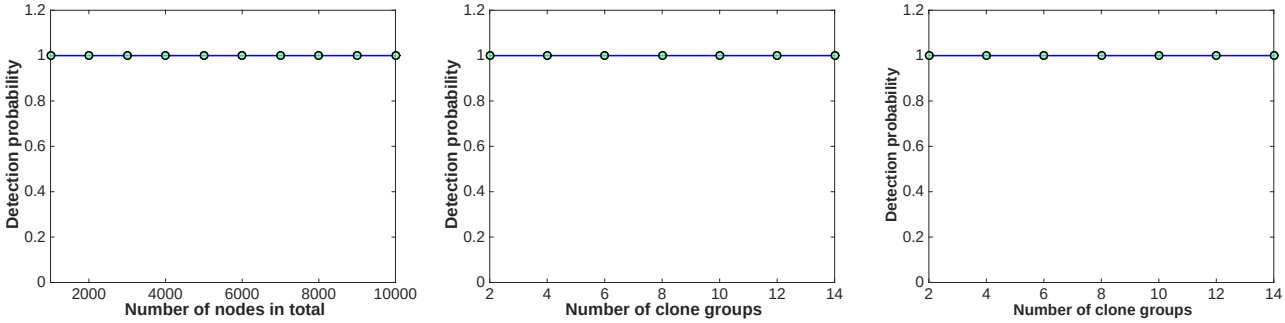
A. Evaluation Results

Because each node in MDSClone only needs to sense the RSS, send out the measured distances from its neighbors, and forward the received neighbor-distance information to the BS, MDSClone in fact only incurs a limited memory overhead. The neighboring information occupies 12029 bytes in ROM, and only 602 bytes in RAM. On the other hand, because each node is assumed to only execute the above steps, some delay will be incurred when a node uses the MDSClone algorithm. Owing to the fact that a node only sends one packet per second (which is our considered setting), the reported detection time will be affected by such a setting. If we ignore the time delay incurred by our hardware setting, we can observe and infer that the computation time on a sensor node is 0.25 seconds. The results of the TOSSIM simulation show that for the MDSClone operations on the mote (i.e., finding the neighboring nodes), the energy consumption due to the use of the microcontroller is 1222 mJ, and the energy consumption due to the use of the radio circuit is 2021 mJ. It is worth noting that the computation of the MDS, which is the main function of our proposed algorithm, is basically performed by the BS, and therefore the computational overhead and energy consumption imposed on the sensor nodes are negligible.

In Figure 3 we report the result of our evaluation of the clone detection probability of MDSClone considering three network settings: (a) varying the total number of nodes in the network from 1,000 to 10,000, while assuming there are two clones in the network; (b) considering a fixed number of nodes in the network, i.e., $n = 1,000$, and varying the number of clones in an ideal network setting without noise; (c) considering a fixed number of nodes, $n = 1,000$, and varying the number of clones, assuming the environment to be noisy due to node mobility, as explained in Section V-A1c. For this experiment we adopted the value of λ that we calculated in Section V-A1a. In practice, the network owner may choose a distortion threshold smaller than the one calculated in Section V-A1a, in order to ensure the successful detection of nearly all clones. Indeed, the choice of a small λ may lead to false positives, i.e., some genuine nodes may be regarded as clones because of distortion due to noisy distance measurements or using the shortest path to approximate the Euclidean distance between two nodes in the MDS calculation. Because the BS may perform attestation on clones instead of having a network-wide revocation of clone IDs, the BS may find that a clone under attestation is genuine node. In this manner, the BS can still almost perfectly detect the clones at the expense of rare false positives.

As we can see in Figure 3, the clone detection probability of MDSClone is 100% in various scenarios. In particular, Figure 3a shows that MDSClone has a full detection probability for both large and small scale networks. Figure 3b depicts the probability of clone detection when varying the number of clone groups. We recall that, as explained in Section III-B, in each *clone group* we considered two clones, for simplicity. In this evaluation, we considered a network of size n=1,000, and we varied the number of clones from two to 14. The reason behind this choice of range is that if we are able to detect the clones when there number in the network is few, it would be much more easier to distinguish them when there are a large number of clones in the network. This follows because from the MDS point of view, more clone groups actually imply more distortion on the reconstructed node map, and therefore they are easier to detect for the BS. Moreover, Figure 3c shows that MDSClone is robust to noisy distance measurements. In particular, as can be seen in the figure, even in the case of $\mathcal{N}(2, 10)$ noise applied to each distance measurement(a Gaussian distribution with mean two and standard deviation two), MDSClone is still able to reconstruct the node map with a slight increase in the approximate preservation distance λ. The approximate preservation of the node map can therefore be used to identify the clones.

It is worth mentioning that some extreme rare cases might still affect the clone detection probability of MDSClone. For example, consider the case in which clones are the same distance from most of the other nodes in the network. For instance, let A and C represent two genuine nodes forming a connection, while A, B, C, and B' form a rectangle, where B and B' are clones sharing the same ID and have the same distance to A and C. If the shape formed by the genuine nodes is symmetric (e.g., the line formed by A and C in the above example), then the attacker can strategically place the two clones (e.g., B and B' in the above example) in such a way that the distances between the genuine nodes and clones are all preserved. In this case, MDSClone fails to detect the clones. In general, the above argument can be extended to the case where the shape formed by genuine nodes satisfies rotational symmetry. If so, an attacker can place a particular number of clones in the network such that the distances between genuine nodes and clones are all preserved, in order to evade detection by MDSClone. However, we argue that such cases occur very rarely in hybrid networks. Considering that (some) nodes in hybrid networks have mobility, the shape formed by genuine



(a) Detection probability vs. number of nodes.

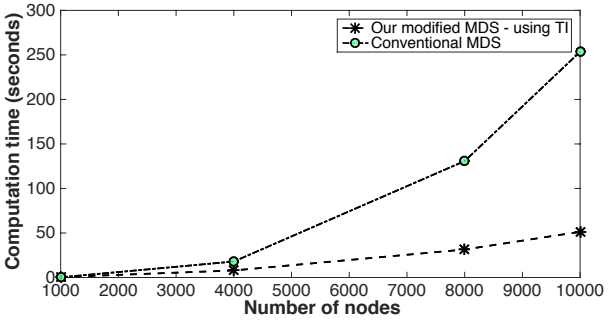
(b) Detection probability vs. number of clone groups (non-noisy environment).

(c) Detection probability vs. number of clone groups (noisy environment).

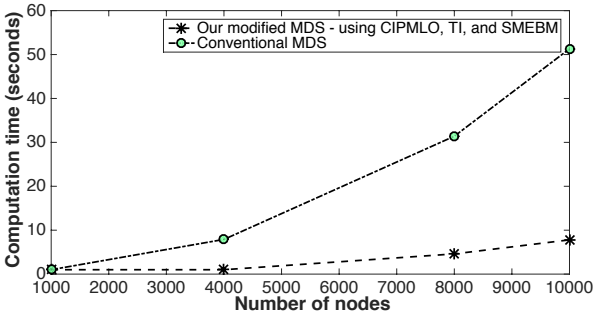
Fig. 3: Clone detection probability.

nodes is constantly changing. At any time slot, the underlying network topology looks like a random geometric graph, and the probability that the shape is symmetric or rotationally symmetric is very low.

Another criterion that we considered in evaluating the performance of MDSClone is the computational time. As is evident in Algorithm 1, the main computational complexity of MDSClone relates to the computing of the MDS function by the BS, and the other operations contribute negligible computational overheads to MDSClone. Therefore, here we just concentrate on the MDS calculation time, and emphasize how our proposed acceleration techniques help to reduce the computational time. More specifically, in Section V-B1 we proposed three techniques to speed up the MDS calculation. In Figure 4a and Figure 4b we compare the computational times for our proposals and the conventional MDS calculation. In particular, in Figure 4a we implemented the *TI* method (explained in Section V-B1b) to considerably speed up the calculation of $B = \Omega C$, which constitutes a significant portion of the computational burden in the MDS calculation. Figure 4a shows that our developed *TI* technique achieves a speed-up by a factor of five compared with the original MDS for a matrix of dimension 10000×10000, simulated in MATLAB and adopting the built-in function `mdscale`. The speed-up gain will be significantly higher when considering a larger matrix. In addition, although the speed-up applies only to the MDS calculation, in fact MDSClone also achieves a speed-up by a factor of five, because MDS constitutes the core part of MDSClone.



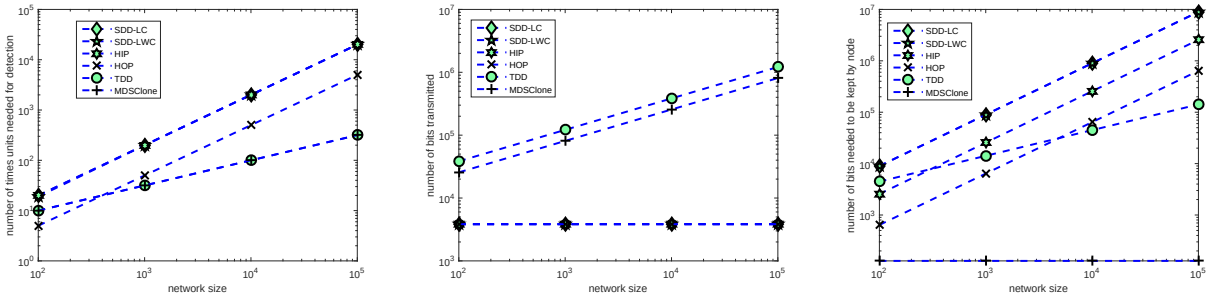
(a) Computational time of $B = \Omega C$ vs. number of nodes.



(b) MDS calculations time vs. number of nodes.

Fig. 4: Analysis of the MDS speed-up proposals.

As can be seen in Figure 4a, the *TI* method calculates $B = \Omega C$ faster than the original MDS algorithm. However, one may have concerns regarding the overall performance improvement of the MDS calculation. Figure 4b shows that by using our developed speed-up techniques (*CIPMLO*, *TI*, and *SMEBM*), we can enhance the speed by more than five times compared with the conventional MDS calculation (e.g., the combined use of a householder transform and eigendecomposition). The reason for the performance discrepancy between Figure 4a and Figure 4b can be attributed to the fact that the speed-up shown in Figure 4b is the result of adopting *CIPMLO*, *TI*, and *SMEBM*, while the speed-up shown in Figure 4a is the result of merely using *TI*. Another important observation from Figure 4b is that considering a network size of 1,000 nodes, our MDS calculation requires around two seconds, and in a very large-scale scenario with 10,000 nodes, it takes less than 10 seconds. It is worth mentioning that in an IoT environment, considering 1,000 nodes in a network is more realistic, and reflects several real-world applications, such as a smart hospital or smart building. At this stage, one may have concerns regarding the total detection time of MDSClone, because Figure 4b shows only one computation of the MDS function, while in MDSClone the BS needs to iteratively perform the MDS calculation in order to identify the clones in the network. It should be noted that because the MDS calculation in each iteration should be performed on a different set of nodes in the network, each iteration is *independent* from the others. Thanks to this *prominent* feature, different iterations of MDS calculation could easily be performed in parallel by several different devices, or powerful servers, leading to a reduction in the computational time of the whole algorithm.



(a) Clone detection time. (b) Communication overhead of a node. (c) Memory overhead of a node.

Fig. 5: Performance comparison between MDSClone and state-of-the-art clone detection methods.

B. Comparative Analysis

In this section, we provide a comparative analysis between MDSClone and the state-of-the-art clone detection methods. For our comparison, we consider the following performance metrics: (i) The amount of time required for clone detection. (ii) The communication overhead at an IoT device. (iii) The memory overhead at an IoT device. We exclude XED [11] and EDD [11], [12] from our comparison, because XED is vulnerable to collusive clones and EDD only works when considering a random waypoint mobility model.

In Figure 5a, we present the required time for clone detection. As can be seen, TDD [13] and MDSClone require considerably less time compared with the other detection methods. Note that TDD and MDSClone are centralized solutions, and therefore each node needs to forward some information (materials for clone detection) to a specific location in the network (e.g., the BS in the case of MDSClone). Because the average hop distance between two nodes in a flat network with n randomly distributed nodes is $O(\sqrt{n})$, with the assumption that each hop distance movement requires one time unit, $O(\sqrt{n})$ time delays will also be considered in TDD and MDSClone. Because each node in distributed protocols such as SDD-LC [13], SDD-LWC [13], and HIP-HOP [5] requires more time to identify witness nodes, distributed detection methods usually incur significantly more delays for clone detection.

Figure 5b shows the communication overhead imposed on each node. As shown in the figure, TDD and MDSClone impose higher communication overheads. This is because of the fact that, similarly to the above case, in these two methods each node needs to forward the required information to a central node. On the other hand, in the case of a distributed solution, each node only needs to communicate with its neighbors. However, it is worth noting that although the communication overhead in MDSClone is seemingly higher than that in distributed solutions, it can be substantially reduced in certain applications. For example, in certain IoT applications (e.g., a smart city) there could be multiple relays capable of cellular communication (e.g., LTE) forming a backbone network with the BS. An IoT device in the proximity of a relay node can forward the neighbor-distance information to the nearby relay, which then forwards the neighbor-distance information back to the BS. In this manner, not only the time, but also the communication cost, can be significantly reduced, while still guaranteeing the detection capability of MDSClone. It is worth noting that the effect of such a scenario on other existing clone detection methods remains unclear.

In addition, we conducted experiments regarding the storage overhead imposed on the IoT nodes. The comparison results are shown in Figure 5c. The storage overhead of MDSClone is close to zero. The reason for this is that in MDSClone each node simply collects neighbor-distance information from neighboring nodes and forwards it to the BS. After forwarding, the node can remove this information from its memory, resulting in a memory footprint of close to zero. However, all of the other detection methods require the IoT device to maintain the historic neighbor-distance information for some time to identify clones, resulting in a considerable memory overhead.

VII. Conclusion

In this paper, we have proposed a clone detection solution, called MDSClone, based on the multidimensional scaling (MDS) algorithm for a heterogeneous IoT environment. We have taken into account the specific features of IoT devices in designing MDSClone, i.e., unawareness of geographical positions, the possibility of being both static and mobile, and the lack of a specific mobility pattern. We showed (in Table I) that compared with the existing clone detection methods, MDSClone provides an outstanding approach, because it is the first method that supports hybrid networks, while its memory cost is of order $O(1)$, its communication cost is affordable, and it is a location-independent method. Moreover, we showed that the clone detection probability of MDSClone is almost 100%, and the MDS calculation algorithm could be parallelized, leading to a shorter detection delay. Therefore, considering all of its advantages, we believe that MDSClone could be considered as a superior candidate for clone detection in real-world IoT scenarios. However, in the case of dense network topologies, our proposal may impose a communication overhead on the network. Therefore, in future work we aim to provide a distributed version of MDSClone for IoT scenarios.

Acknowledgment

Chia-Mu Yu is partially funded by MOST 106-3114-E-005-001, MOST 106-2218-E-155-007, MOST 106-2221-E-005-017, MOST 105-2923-E-001-002-MY2, MOST 105-2923-E-002-014-MY3, MOST 105-2218-E-155-010, and MOST 104-2628-E-155-001-MY2. Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980), as well as the EU TagItSmart!

Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896), the CNR-MOST/Taiwan 2016-17 project "Verifiable Data Structure Streaming", the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation, and the grant "Scalable IoT Management and Key security aspects in 5G systems" from Intel. This work is also partially supported by the EU H2020 SYMBIOTE Project (agreement 688156).

REFERENCES

[1] S. Gaur, "Bringing context awareness to iot-based wireless sensor networks," in *PerCom'15*. IEEE, 2015.

[2] A. Solanas, C. Patsakis, M. Conti, I. Vlachos, V. Ramos, F. Falcone, O. Postolache, P. Perez-martinez, R. Di Pietro, D. Perrea, and A. Martnez-Balleste, "Smart health: a context-aware health paradigm within smart cities," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 74–81, 2014.

[3] Y. Wang, G. Attebury, and B. Ramamurthy, "A survey of security issues in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 2, pp. 2–23.

[4] O. Garcia-Morchon, S. Keoh, S. Kumar, R. Hummen, and R. Struik, "Security considerations in the ip-based internet of things," 2012. [Online]. Available: https://tools.ietf.org/html/draft-garcia-core-security-04

[5] M. Conti, R. Di Pietro, and A. Spognardi, "Clone wars: Distributed detection of clone attacks in mobile wsns," *Journal of Computer and System Sciences*, vol. 80, no. 3, pp. 654–669, 2014.

[6] M. Conti, "Clone detection," in *Secure Wireless Sensor Networks*. Springer, 2016, pp. 75–100.

[7] A. K. Mishra and A. K. Turuk, "A comparative analysis of node replica detection schemes in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 61, pp. 21–32, 2016.

[8] W. T. Zhu, J. Zhou, R. H. Deng, and F. Bao, "Detecting node replication attacks in wireless sensor networks: a survey," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 1022–1034, 2012.

[9] Z. Chen, F. Xia, T. Huang, F. Bu, and H. Wang, "A localization method for the internet of things," *The Journal of Supercomputing*, pp. 1–18, 2013.

[10] O. Bello and S. Zeadally, "Intelligent device-to-device communication in the internet of things," *IEEE Systems Journal*, vol. 10, no. 3, pp. 1172–1182, 2016.

[11] C.-M. Yu, Y.-T. Tsou, C.-S. Lu, and S.-Y. Kuo, "Localized algorithms for detection of node replication attacks in mobile sensor networks," *IEEE Transactions on Information Forensics and Security,*, vol. 8, no. 5, pp. 754–768, 2013.

[12] C.-M. Yu, C.-S. Lu, and S.-Y. Kuo, "Efficient and distributed detection of node replication attacks in mobile sensor networks," in *VTC'09*. IEEE, 2009.

[13] K. Xing and X. Cheng, "From time domain to space domain: Detecting replica attacks in mobile ad hoc networks," in *INFOCOM'10*, 2010.

[14] J. B. Kruskal and M. Wish, *Multidimensional scaling*. Sage, 1978, vol. 11.

[15] Y. Shang, W. Ruml, Y. Zhang, and M. P. Fromherz, "Localization from mere connectivity," in *MobiHoc'03*. ACM, 2003, pp. 201–212.

[16] Narrow Band Internet of Things (NB-IoT). [Online]. Available: http://www.gsma.com/connectedliving/narrow-band-internet-of-things-nb-iot/

[17] LoRa Alliance - Wide Area Networks for IoT. [Online]. Available: https://www.lora-alliance.org/

[18] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *IEEE Symposium on Security and Privacy*. IEEE, 2005, pp. 49–63.

[19] B. Zhu, S. Setia, S. Jajodia, S. Roy, and L. Wang, "Localized multicast: efficient and distributed replica detection in large-scale sensor networks," *IEEE Transaction on Mobile Computing*, vol. 9, no. 7, pp. 913–926, 2010.

[20] M. Conti, R. D. Pietro, L. V. Mancini, and A. Mei, "Distributed detection of clone attacks in wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 685–698, 2011.

[21] M. Zhang, V. Khanapure, S. Chen, and X. Xiao, "Memory efficient protocols for detecting node replication attacks in wireless sensor networks," in *ICNP' 09*. IEEE, 2009, pp. 284–293.

[22] M. Dong, K. Ota, L. T. Yang, A. Liu, and M. Guo, "Lscd: A low-storage clone detection protocol for cyber-physical systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 712–723, 2016.

[23] K. Xing, F. Liu, X. Cheng, and D. H. Du, "Real-time detection of clone attacks in wireless sensor networks," in *ICDCS'08*. IEEE, 2008, pp. 3–10.

[24] R. Brooks, P. Govindaraju, M. Pirretti, N. Vijaykrishnan, and M. T. Kandemir, "On the detection of clones in sensor networks using random key predistribution," *IEEE Transactions on SMC, Part C*, vol. 37, no. 6, pp. 1246–1258, 2007.

[25] H. Choi, S. Zhu, and T. F. La Porta, "Set: Detecting node clones in sensor networks," in *SecureComm'07*. IEEE, 2007, pp. 341–350.

[26] J.-W. Ho, M. Wright, and S. K. Das, "Fast detection of mobile replica node attacks in wireless sensor networks using sequential hypothesis testing," in *IEEE Transactions on Mobile Computing*, vol. 10, no. 6. IEEE, 2011, pp. 767–782.

[27] Y. Zeng, J. Cao, S. Zhang, S. Guo, and L. Xie, "Random-walk based approach to detect clone attacks in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 5, pp. 677–691, 2010.

[28] Z. Zheng, A. Liu, L. X. Cai, Z. Chen, and S. X. (Shermen), "Ercd: An energy-efficient clone detection protocol in wsns," in *INFOCOM'13*. IEEE, 2013.

[29] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[30] A. Yassin, Y. Nasser, M. Awad, A. Al-Dubai, R. Liu, C. Yuen, R. Raulefs, and E. Aboutanios, "Recent advances in indoor localization: A survey on theoretical approaches and applications," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 1327–1346, 2016.

[31] M. Conti, R. Di Pietro, L. V. Mancini, and A. Mei, "A randomized, efficient, and distributed protocol for the detection of node replication attacks in wireless sensor networks," in *MobiHoc'07*. ACM, 2007, pp. 80–89.

[32] S. Sciancalepore, G. Piro, G. Boggia, and G. Bianchi, "Public key authentication and key agreement in iot devices with minimal airtime consumption," *IEEE Embedded Systems Letters*, vol. 9, no. 1, pp. 1–4, 2017.

[33] G. Ateniese, G. Bianchi, A. Capossele, and C. Petrioli, "Low-cost standard signatures in wireless sensor networks: a case for reviving pre-computation techniques?" in *NDSS'13*, 2013.

[34] P. Szalachowski and A. Perrig, "Lightweight protection of group content distribution," in *IoTPTS'15*. ACM, 2015, pp. 35–42.

[35] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *S&P'04*. IEEE, 2004, pp. 272–282.

[36] S. Du, N. Zheng, S. Ying, and J. Liu, "Affine iterative closest point algorithm for point set registration," *Pattern Recognition Letters*, vol. 31, no. 9, pp. 791–799, 2010.

[37] P. J. Besl, N. D. McKay *et al.*, "A method for registration of 3-d shapes," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[38] M. Greenspan and M. Yurick, "Approximate kd tree search for efficient icp," in *3DIM'03*. IEEE, 2003, pp. 442–448.

[39] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, 1996.

[40] A. S. Householder, *The theory of matrices in numerical analysis*. Courier Corporation, 2013.

[41] EISPACK, eigenvalue computation. [Online]. Available: http://www.netlib.org/eispack/

[42] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in *SenSys'03*. ACM, 2003, pp. 126–137.