Decentralized Robot Swarm Clustering: Adding Resilience to Malicious Masquerade Attacks

Mitali Gandhe[1] and Michael Otte[2]

[1] Georgia Institute of Technology, Atlanta, GA 30332
[2] University of Maryland, College Park, MD 20742

Abstract. We compare the resilience of four distributed robot swarm clustering algorithms to masquerade attacks launched from malicious robots within the swarm. The clustering algorithms are distributed variants of DBSCAN and k-Means that have been modified for use on a distributed robot swarm that only has access to *local* communication and *local* distance measurements. We subject these distributed variants of k-Means and DBSCAN to malicious masquerade attacks and observe how clustering performance is affected. We then modify each variant to include a distributed Intrusion Detection and Response System (IDRS) to detect malicious robots and maintain the swarm's integrity despite an attack. We evaluate all four variants both in simulation and in a hardware testbed containing a swarm of 25 Kilobot robots. We find that centralizing data within the swarm makes the swarm more vulnerable to malicious attacks, and that distributed IDRS relying on local message passing can effectively identify malicious robots and reduce their negative effects on swarm clustering performance.

1 Introduction

Swarm robotics is a rapidly developing field with far-ranging applications including disaster relief, environmental monitoring, and defense [6]. In such a distributed network, individual robots act upon their perceived environment and incoming communication, which present vulnerabilities that malicious attackers may exploit. Therefore, developing security systems is crucial to ensuring safe real-world applications of swarms [8, 16].

Swarm algorithms are typically *decentralized* to eliminate single points of failure and *distributed* to leverage the sensors, actuators, and computation across many robots. The hardware swarms may contain many robots (10s, 100s, 1000s, etc.), and so robot swarm algorithms should be *scalable*, retaining functionality even if the number of robots in the swarm changes by orders of magnitude. *Local communication* involves message passing between neighboring robots, whereas *global communication* allows message passing from any robot to any other robot. Most swarm algorithms use local communication to avoid communication bandwidth saturation as swarm size increases. Though a global positioning system (GPS) can use either local or global communication, many important problem

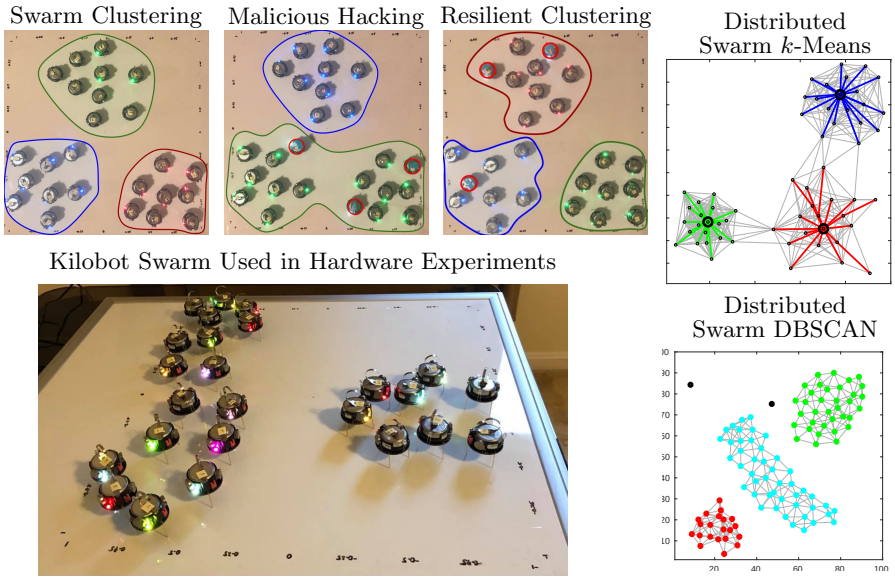

Fig. 1. **Top-Left:** We evaluate the resilience of distributed swarm clustering algorithms to malicious masquerade attacks (center), and implement security measures to protect the swarm (right). Color overlays indicate robot clusters, and malicious robots are outlined with red circles. **Right:** Variants of the k-Means (Left) and DBSCAN (Right) algorithms modified for use by swarms using local communication and positioning. **Bottom-Left:** Hardware experiments are run on a swarm of 25 Kilobots.

domains lack GPS data, e.g., environments that are underground, underwater, or extraterrestrial—as well as GPS-denied adversarial scenarios.

We develop and analyze two distributed swarm *clustering* algorithms. Clustering is the process of dividing objects into groups based on like qualities, where objects with similar *features* are placed in the same cluster and objects with dissimilar features are placed into different clusters [19]. Distributed clustering algorithms allow the swarm to determine position and orientation data by establishing a local coordinate system [16]. Clustering a swarm's robots into groups based on proximity helps analyze the swarm's topology, and may serve as a pre-processing step for swarm behaviors such as sub-team formation, task division, geometric analysis, data aggregation, and information distribution.

However, the distributed process of clustering robots into groups presents an opportunity for a malicious robot to hijack the swarm (Figure 1: top left). If a robot was tampered with before programming or captured and returned to the swarm, existing security such as encryption may not protect the swarm [2]. As a result, the swarm may be vulnerable to a masquerade attack: an insider attack intended to disrupt the outcome of the algorithm. We develop masquerade attacks that exploit weaknesses of the two distributed clustering algorithms.

To discuss the security of robot swarms, we borrow inspiration and vocabulary from the well-studied field of security in decentralized networks. An *Intrusion Detection System* (IDS) identifies adversaries and an *Intrusion Response System* (IRS) counteracts the effect of the adversaries [6, 17]. An *Intrusion Detection and Response System* (IDRS) does both.

In this paper we adapt two clustering algorithms, k-Means and DBSCAN (density-based spatial clustering of applications with noise), to suit a decentralized robot swarm operating in scenarios *without* **global communication and** *without* **global position data. Next, we design masquerade attacks that take advantage of natural vulnerabilities in these algorithms. Finally, we create new variants of the algorithms that have IDRS designed to detect such attacks. The main contribution of this paper is an evaluation of whether or not—and to what extent—a decentralized IDRS can improve robot swarm clustering performance by detecting and neutralizing malicious robots.**

This paper is organized as follows. Section 2 discusses related work. Section 3 outlines notation and formal problem statements. Sections 4 and 5 present distributed swarm clustering algorithms (DBSCAN and k-Means, respectively), masquerade attacks against them, and resilient variants with IDRS. Section 6 describes experiments run in hardware testbeds and in simulation to evaluate the algorithms' resilience to the masquerade attacks. Section 7 details our results and Section 8 contains our conclusion.

2 Related Work

The originial k-Means [10] and DBSCAN [9] algorithms were developed in the 1960s and 1970s. Recent variants take advantage of distributed computing [11, 5] including over peer-to-peer, wireless, and ad hoc networks [21, 20]. Work has been done to ensure the privacy of the data being clustered by such algorithms [13, 1]. Existing distributed approaches seek to leverage the computation of multiple computers while minimizing communication between them. In contrast, we explore the distributed *swarm* clustering problem, in which each robot administers a single data point—its own location—and clustering is the result of an emergent process that uses continual local message passing between neighboring robots. As a result, our algorithms are implemented to suit swarm-specific constraints, including message dropping, finite bandwidth, restricted space overhead, and limited computational power.

McCune and Madey perform simulations of an ant-inspired pick-up, carry, and drop algorithm for resource aggregation they call "Decentralized k-Means Clustering" drawing a parallel between k base stations (cluster centers/ant nest), mobile agents (transport nodes/ants), and stationary sensors (resources, food) [12]. The contrasting metaphor that applies to our work is that robot locations are data points—and the objective is to find k cluster centers (also robots) that divide the swarm into proximity based sub-teams.

The k-Medians algorithm is closely related to k-Means [18, 3]. Previous work has defined the distinction between the two as follows: k-Means minimizes the sum of L^2 norms over all nodes to their corresponding cluster's center, whereas k-Medians minimizes the corresponding sum of L^1 norms. In contrast, the swarm algorithms that we investigate use the graph distance metric as defined as summed distance through/along the communication network (and not through the Euclidean space in which the graph is embedded). The graph distance metric is useful for robot swarms because it can be calculated using only local communication and local distance measurements.

Existing work also identifies threats to the swarm [6, 7]. In one example, Gil *et al.* use radio signatures to detect when an adversarial robot generates fake identities to gain influence within the swarm, a technique known as *spoofing* [4]. They show that robots equipped with appropriate sensors can help identify malicious robots by their hardware signature. We differentiate the spoofing attacks Gil *et al.* study (in which an adversary generates fake *identities*) from masquerade attacks (in which an adversary generates fake *data*), and develop an intrusion detection system to identify the latter.

In contrast to past work on distributed clustering algorithms, our research focuses on how adversarial robots can influence the clustering algorithm's behavior, and to what extent a distributed IDRS can provide resilience to malicious actors. A preliminary non-archival poster of this work was presented at the IEEE International Symposium on Multi-Robot and Multi-Agent Systems in 2021.

3 Notation & Problem Statements

The robot swarm $S = \{r_1, \ldots, r_n\}$ contains n robots with unique IDs $1 \ldots n$. Let d_{ij} denote the distance between robots r_i and r_j. We assume that if robot r_i can communicate with r_j then r_j can determine d_{ij}. We do **not** require robots to know their global positions. The distance between two neighboring robots can be determined without global positions, for example, by observing message time of flight or received signal strength. The communication graph over the swarm is $G = (V, E)$, where we abuse our notation by letting robots represent their (own) respective nodes in the node set $V \equiv S = \{r_1, \ldots, r_n\}$. The existence of a directed edge $(r_i, r_j) \in E$ indicates that robot r_i can communicate with r_j.

While communication between robots in the real world is generally nonsymmetric, it is algorithmically convenient to impose symmetry in the communication graph such that $(r_i, r_j) \in E \iff (r_j, r_i) \in E$. This can be done by having robot r_j drop messages received from robot r_i whenever d_{ij} is greater than a user defined threshold distance d, i.e., such that the human user knows a swarm's communication hardware will reliably send/receive messages closer than d. Let $d_{\max}$ be the maximum distance a robot can ensure bidirectional communication. Thus, a symmetric d-disc communication graph can be achieved by having robot r_j accept messages from robot r_i only if $d_{ij} = d_{ji} \leq d$ for $d \leq d_{\max}$.

Let $E_d = \{(i,j) \,|\, d_{ij} \leq d\}$ be the set of all edges of length d or less. Define $G_d = (V, E_d)$. Having robots drop messages as described above, guarantees

$$((r_i, r_j) \in E_d) \wedge (d \leq d_{\max}) \implies ((r_j, r_i) \in E_d)$$

In other words, robot r_j can infer that if it accepts a message from r_i then the sender will accept messages from r_j in return. Cluster membership is determined based on the topology of the d-disc communication graph G_d.

3.1 Attack & IDRS Model

Existing work defines a masquerade attack as an insider attack that generates fake data to target specific weakness of an algorithm [2]. By tampering with a member of the swarm or intercepting messages, a malicious agent can infer basic information about the algorithm structure and message content. We design attacks to disrupt the swarm's emergent behavior in the simplest and most effective manner possible. These attacks are intended to simulate more sophisticated attack models.

Similarly, our IDRS is a prototype for more sophisticated *anomaly-based* intrusion detection. Higgins *et al.* note that responding to masquerade attacks often requires IDRS tailored to the specific mechanism of attack [6]. Therefore, we implement the simplest effective IDRS that will counter the attack, in order to analyze how an intrusion detection system interacts with the emergent behavior of robotic swarms. The attack and IDRS functionalities are encapsulated in the code description to capture this idea.

3.2 Formal Problem Statements

We now define the swarm clustering and hacking problems.

Problem 1. Distributed Swarm Clustering: *Given a swarm of n robots that communicate locally, the swarm must collectively divide itself into mutually exclusive clusters based on robot's relative proximity such that robots within a particular cluster are closer to other robots in their own cluster then they are to robots in the other clusters.*

Problem 2. Swarm Clustering Masquerade Attack: *Given a swarm of n robots that communicate locally and that is attempting to solve Problem 1, one or more malicious robot(s) must cause the swarm to find a lower quality solution (or prevent the swarm from finding any solution) by injecting incorrect data into the distributed algorithm–but not by simply disrupting communication.* A notable element of Problem 2 is that the malicious robots seek to alter the outcome of the swarm algorithm instead of simply blocking communication.

Problem 3. Resilient Swarm Clustering: *Given a swarm of n robots that communicate locally and that are solving Problem 1, as well as one or more malicious robots that are attempting to hack the solution by solving Problem 2, the swarm must identify and neutralize the malicious robots.*

4 Distributed Swarm DBSCAN, Attack, and IDRS

The original (centralized) DBSCAN algorithm has two parameters: a distance threshold d that determines whether or not two nodes are neighbors and the minimum number of neighbors m a node must have to be considered an 'internal' node. The algorithm creates a d-disc graph, such that a particular node's neighbor set $\mathbf{N}$ contains all nodes within distance d of that node. Nodes are defined as being 'internal' if $|\mathbf{N}| \geq m$ neighbors, 'boundary' if $m > |\mathbf{N}| \geq 1$, and 'outlier' if $|\mathbf{N}| = 0$. DBSCAN defines that neighboring internal nodes are in the same cluster, boundary nodes are allowed to join any one of their neighbor's clusters, and outlier nodes are not considered to be in any cluster.

The fact that DBSCAN uses the topology of a d-disc graph to perform clustering makes it well suited to distributed implementation on a robot swarm with local communication. For any physically defined communication radius $d_{\max}$, it is possible to run distributed swarm DBSCAN for any $d \leq d_{\max}$. After nodes (robots) have determined if they are internal, boundary, or outlier, the internal robots of each cluster run a distributed consensus algorithm to agree on a cluster ID. This is accomplished by having internal nodes iteratively exchange and average a cluster ID number with their neighboring internal nodes.

We explore a DBSCAN masquerade attack in which multiple malicious robots, dispersed throughout the swarm, cause different clusters to converge to the same value. Each malicious robot advertises itself as an internal robot and then injects data designed to hijack the consensus algorithm (for example, repeatedly broadcasting $-\infty$ or 0). If two or more adversarial robots are located in different clusters, then each cluster containing an adversarial robot can be tricked to converge to the same label. The overall effect is that different clusters that should be considered unique erroneously believe that they are part of a single larger cluster. A graphical depiction of this attack appears in Figure 2.

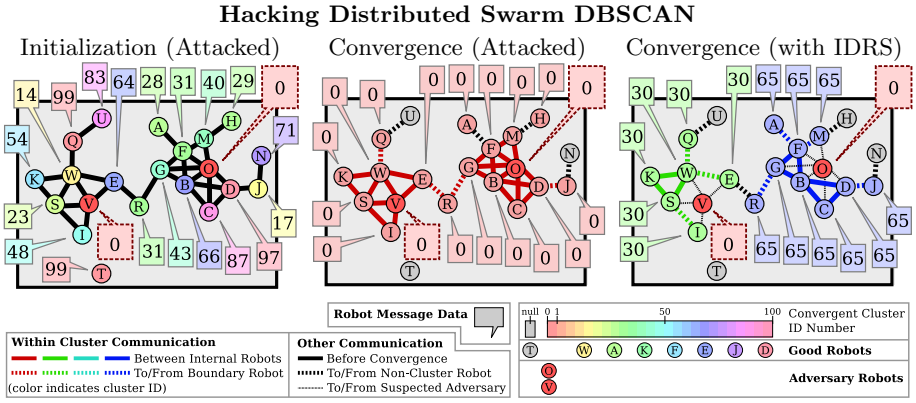

Fig. 2. A masquerade attack on Distributed Swarm DBSCAN. Multiple malicious robots cause different clusters to erroneously converge to the same cluster label.

To detect and overcome this masquerade attack, we augment Distributed Swarm DBSCAN with a distributed IDRS. We call the resulting algorithm variant Resilient Swarm DBSCAN. The IDRS works by having each node maintain lists of suspected malicious actors $\mathbf{B}$—defined as nodes that have not updated their convergent values in more than $y_{\max}$ communications. Suspected bad actors are removed from neighbor lists so that they cannot influence the distributed consensus algorithm. Each robot maintains its own list of suspected malicious robots. To isolate an malicious robot from the swarm, all neighboring robots must detect it separately (by observing unchanging cluster values). A robot that has been incorrectly labeled as malicious by one robot can still partially contribute to the swarm's clustering by communicating through other neighbors.

A main goal of our research is to evaluate *the extent to which using a distributed IDRS can increase a swarm's resilience to malicious attacks—assuming that malicious robots can be detected* (perhaps with occasional false positives). The simple masquerade attack and IDRS described above are useful because they achieve this goal. If malicious robots use more sophisticated attacks (for example, sending random numbers), then more sophisticated IDRS could (and should) be used instead. In our presentation of Resilient Swarm DBSCAN, all IDRS functionality is consolidated in two subroutines: `recordBadActorStatistics`() and `calculateBadActors`(). More sophisticated IDRS can be incorporated into the Distributed Swarm DBSCAN algorithm by adding additional functionality to these two subroutines.

Algorithm 1 Distributed (Resilient) Swarm DBSCAN

Require: i, d, m, $y_{\max}$
1: `initDbscan`() /∗ initializes variables ∗/
2: **loop**
3: **if** received new message M **then**
4: $j \leftarrow M.senderID$
5: **if** $d_{ji} > d$ **or** $j \in \mathbf{B}$ **then**
/∗ out of neighborhood radius ∗/
/∗ or from a suspected malicious ∗/
6: **continue**
7: $\mathbf{N} \leftarrow \mathbf{N} \cup \{j\}$
8: **if** $M.status = internal$ **then**
9: $\mathbf{N}_{\text{int}} \leftarrow \mathbf{N}_{\text{int}} \cup \{j\}$
10: `recordBadActorStatistics`(M)
11: $C_j \leftarrow M.C$
12: $\mathbf{B} \leftarrow$ `calculateBadActors`()
13: `blockBadActors`()
14: `populateMessageMetaData`($\tilde{M}$)
15: **if** $|\mathbf{N}| \geq m$ **then**
/∗ this node i is an internal node ∗/
16: $C_i \leftarrow \texttt{integer}\left(\left(C_i + \sum_{j \in \mathbf{N}_{\text{int}}} C_j\right)/(1 + |\mathbf{N}_{\text{int}}|)\right)$
17: $\tilde{M}.status \leftarrow internal$
18: **else if** $|\mathbf{N}| \geq 1$ **then**
/∗ this node i is a boundary node ∗/
19: **if** $\mathbf{N}_{\text{int}} \neq \emptyset$ **then**
20: $j \leftarrow$ closest member of $\mathbf{N}_{\text{int}}$
21: $C_i \leftarrow C_j$
22: $\tilde{M}.status \leftarrow bouandary$
23: **else**
/∗ this node i is an outlier node ∗/
24: $\tilde{M}.status \leftarrow outlier$
25: `Broadcast`($\tilde{M}$)

Algorithm 2 `initDbscan`()

1: $\mathbf{N} \leftarrow \emptyset$ /∗ this node's neighbors ∗/
2: $\mathbf{N}_{\text{int}} \leftarrow \emptyset$ /∗ this node's internal neighbors ∗/
3: $C_i \leftarrow$ `randomInteger`() /∗ random cluster number ∗/
4: $\mathbf{B} \leftarrow \emptyset$ /∗ list of bad actors ∗/
5: $\mathbf{D}^{\text{same}} \leftarrow \{0, \ldots, 0\}$ /∗ count duplicate sends ∗/

Algorithm 3 `populateMessageMetaData`($\tilde{M}$)

1: $\tilde{M}.senderID \leftarrow i$
2: $\tilde{M}.C \leftarrow C_i$

Lines & subroutines in blue (including those below) are only used in Resilient Swarm DBSCAN, the algorithm variant that incorporates an IDRS.

Algorithm 4 `recordBadActorStatistics`(M)

1: **if** $C_j \neq M.C$ **then**
2: $\mathbf{D}_{\mathbf{j}}^{\text{same}} \leftarrow 0$
3: **else if** $C_j \neq C_i$ **then**
4: $\mathbf{D}_{\mathbf{j}}^{\text{same}} \leftarrow \mathbf{D}_{\mathbf{j}}^{\text{same}} + 1$

Algorithm 5 `calculateBadActors`()

1: **for** $j \in \mathbf{N}_{\text{int}}$ **do**
2: **if** $\mathbf{D}_{\mathbf{j}}^{\text{same}} > y_{\max}$ **then**
3: $\mathbf{B} \leftarrow \mathbf{B} \cup \{j\}$
4: **return B**

Algorithm 6 `blockBadActors`()

1: $\mathbf{N} \leftarrow \mathbf{N} \setminus \mathbf{B}$
2: $\mathbf{N}_{\text{int}} \leftarrow \mathbf{N}_{\text{int}} \setminus \mathbf{B}$

Pseudocode for the Swarm DBSCAN algorithms appears in Algorithm 1. Lines involved with the IDRS are colored blue. Distributed Swarm DBSCAN (without IDRS) is described using only the black lines. Resilient Swarm DBSCAN (with IDRS) is described using black and blue lines.

The algorithm requires input parameters d (neighborhood distance), m (number of neighbors required by internal nodes), and unique robot ID i. Initialization is performed by the subroutine `initDbscan`(); this node's parent, neighbor set $\mathbf{N}$, and internal neighbor set $\mathbf{N}_{\text{int}}$ are initialized to empty, and C_i is drawn as a random number. While $\mathbf{N}$ will eventually contain all neighbors within communication range, $\mathbf{N}_{\text{int}}$ is the set of neighbors that are also internal nodes.

Message passing is used to transfer data throughout the swarm. Messages are received (lines 3-4) and then ignored if they are sent from robots beyond the neighborhood distance threshold (lines 5 and 6). Newly discovered neighbors are added to this node's neighbor set (line 7), internal neighbors are added to this node's internal neighbor set (lines 8-9), and the sending neighbors continually converging cluster ID number C_j is updated (line 11). In the second half of the algorithm this robot's data is shared with neighbors (lines 14-25). Message data (this robot's ID i and current cluster ID integer C_i) is populated using the subroutine `populateMessageMetaData`() (line 14). The message is populated with this node's current internal, boundary, or outlier status (lines 17, 22, and 24). If this node is an internal node, then it calculates the new value of its converging cluster ID (line 16), and if it is a boundary node, then it selects the cluster of its closest neighbor to join (line 20).

Several changes implement the IDRS in the Resilient DBSCAN algorithm. The message data is processed to calculate suspected malicious robots (lines 10 and 12), messages are rejected from suspicious robots (lines 5-6), and suspicious robots are removed form neighbor lists via the subroutine via the subroutine `blockBadActors`() (line 13).

5 Distributed Swarm *k*-Means: Algorithm, Attack, IDRS

The distributed swarm k-Means algorithm is designed for use with a robot swarm that uses (only) local communication and local distance information. Each of the k clusters are associated with a cluster "root" robot. Cluster roots are analogous to the cluster centers used in the original k-Means algorithm. Cluster membership of each robot is determined by the closest root robot. At the beginning of the algorithm the initial k root robots can be chosen randomly (for unsupervised clustering) or selected by a user (for semi-supervised clustering).

The algorithm is iterative at the swarm level. Each swarm level iteration involves running the following two distributed processes:

1. **Determine the current cluster membership of all robots**. This is accomplished using a multi-goal distributed Dijkstra's reverse search to create a shortest spanning forest that contains k trees—one per cluster. Each cluster's "root" robot is defined to be a Dijkstra's "goal" and so becomes a root of one of the forest's k different shortest-path trees. After this distributed

calculation, each robot knows its shortest-path graph distance to the nearest root robot. Each robot adopts the cluster label of its tree's root.

2. **Transfer each cluster's root toward the middle of that cluster**. A consequence of using (only) local communication is that the responsibility of being a cluster's root must pass from neighbor to neighbor—instead of jumping directly to the cluster's centroid as in a standard implementation of k-Means. First, all nodes participate in a distributed calculation that provides each node with the size of its current sub-tree and those of its children. Next, each of the k cluster roots determines which of its own children has the largest sub-tree, and selects that child to become its cluster's new root. This causes the cluster root to move one hop toward the cluster's middle.

The two step process (calculating cluster membership and then passing each cluster's roots one communication hop toward its middle) resembles a form of gradient ascent (see Figures 3 and 4). Passing the root from robot to robot makes sense when a swarm has only local communication. However, it has the consequence that the root location will settle into small cycles in the vicinity of local optima. A separate stopping criterion is required to ensure the algorithm will halt. The user parameter $z_{\max}$ defines the maximum number of times any node may pass root responsibility to another robot. After a robot has been root $z_{\max} + 1$ times it retains root responsibility and the algorithm converges.

A malicious robot can trick the swarm into making it a root by falsely advertising a large sub-tree size. If the malicious robot is a neighbor of a cluster root, then advertising a large sub-tree size will ensure that it becomes a root for the next iteration. If the the malicious robot is not a direct neighbor of a cluster root, then well intentioned parents, grandparents, etc. of the malicious robot will incorporate the malicious robot's incorrectly large sub-tree size into their own sub-tree size calculations. Thus, root status will transfer one step closer to

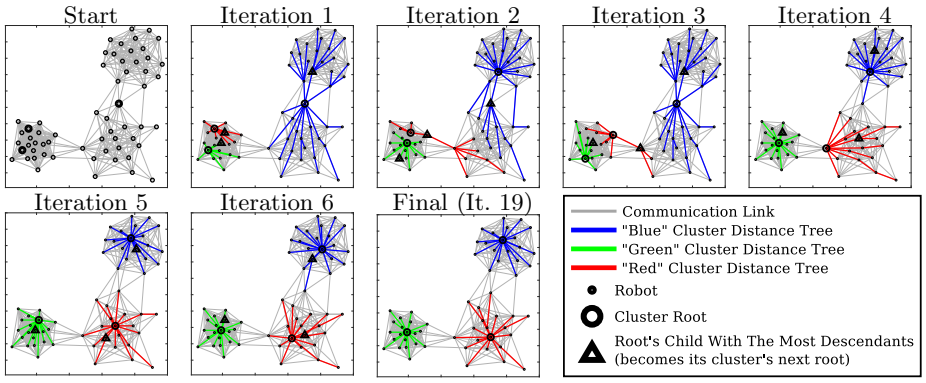

Fig. 3. Example of the distributed swarm k-Means algorithm when $k = 3$. The algorithm is designed for a swarm of robots using local communication and range sensors to calculate distances to neighbors and determine cluster centers.

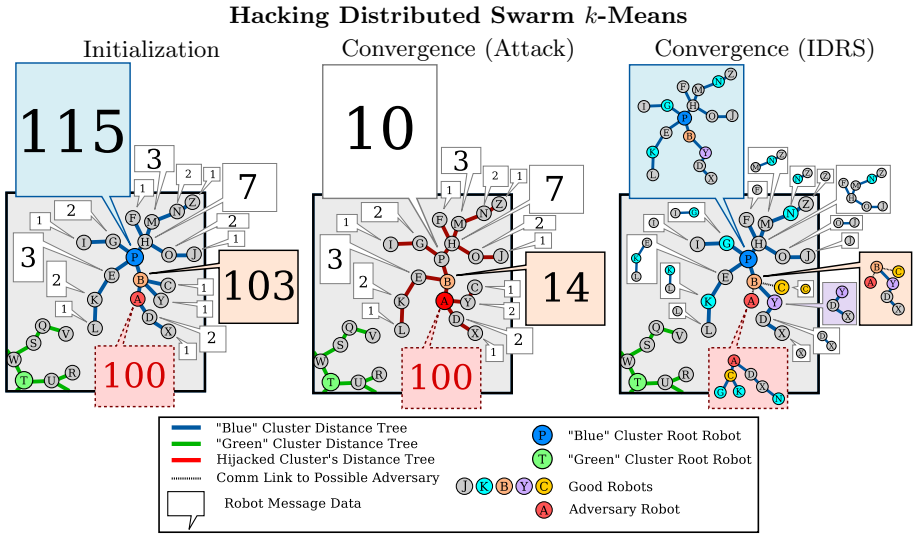

Fig. 4. Root status passes to the root's child with the largest sub-tree (left, blue cluster will pass from node 'P' to node 'B'). A malicious robot can hijack the swarm by advertising fake large sub-tree size (center). In a distributed IDRS that detects this attack robots send the IDs of nodes in their sub-trees. Robots suspected of being malicious—due to conflicting sub-tree information—are ignored (right).

the malicious robot each iteration. Eventually, a direct neighbor of the malicious robot will become root (by induction), and then the malicious robot itself.

A distributed IDRS can be added to Distributed Swarm k-Means to create a new algorithm variant we call Resilient Swarm k-Means. The distributed IDRS works by having each robot send the IDs of all nodes in its sub-trees (instead of only sending its sub-tree size). In a spanning forest, each node can only appear in one branch of a single tree. If two robots on different branches claim to have the same robot as part of their sub-tree, then the other robots add both (potentially malicious) robots to a malicious robot list. This reduces the damage a single malicious robot can cause.

All robots share their lists and robots suspected of being malicious are not included in the next shortest spanning forest. This provides a means of detecting *some* good robots that have been accidentally labeled as malicious. Because the new shortest spanning forest will ignore (and exclude) malicious robots, then robots that broadcast sub-trees containing only themselves cannot be malicious.

This particular IDRS requires an increase in message size from $O(1)$ to $O(n)$, which may not be appropriate in all cases. As with the DBSCAN attack and IDRS described in the previous section, our main goal is to evaluate how algorithmic performance changes with and without the use of IDRS. We do not claim that this is the best possible IDRS—only that it enables the swarm to detect the masquerade attack described above. This is useful for understanding the potential benefits of using swarm IDRS more generally. More sophisticated attacks

Algorithm 7 Distributed (Resilient) Swarm k-Means

Require: i, $\mathbf{C}_{\text{init}}$, $z_{\max}$, $T_{\min}$
1: `initializeKmeans()` /* initializes variables */
2: **loop**
3: **if** received new message M **then**
4: $j \leftarrow M.senderID$
5: **if** $M.t < t$ **or** $d_{ji} > d$ **or** $j \in \mathbf{B}$ **then**
/* message out of date or beyond radius */
/* or from suspected malicious */
6: **continue**
7: **if** $M.t > t$ **then**
/* a new k-Means iteration has started */
8: `resetDistanceTree()`
9: `conflictDurationReset`$(\mathbf{N}_c \setminus \mathbf{B}, \mathbf{N}_c)$
10: $t \leftarrow M.t$
11: `updateDistanceTree`(M)
12: $\mathbf{B} \leftarrow$ `calculateBadActors()`
13: $\mathbf{B} \leftarrow$ `calculateGoodActors()`
14: `blockBadActors()`
15: $\mathbf{S}_i \leftarrow \{i\} + \bigcup_{j \in \mathbf{N}_c} \mathbf{S}_j$
16: `populateMessageMetaData`$(\tilde{M})$
17: $\tilde{M}.\mathbf{S} \leftarrow \mathbf{S}_i$
18: $\tilde{M}.\mathbf{B} \leftarrow \mathbf{B}$
19: **if** $d_{\text{root}} = 0$ **and** $z \leq z_{\max}$ **and** `currentTime()` $- lastUpdated \geq T_{\min}$ **then**
/* pass root to child with largest sub-tree */
20: $\tilde{M}.newRootID = \arg\max_{j \in \mathbf{N}_c} |\mathbf{S}_j|$
21: `Broadcast`$(\tilde{M})$

Algorithm 8 `initializeKmeans()`

1: $parent \leftarrow \emptyset$ /* the parent of this node */
2: $t \leftarrow 0$ /* k-Means iteration */
3: $d_{\text{root}} \leftarrow \infty$ /* distance to root */
4: $\mathbf{N}_c \leftarrow \emptyset$ /* set of this node's children */
5: $z \leftarrow 0$ /* times this robot has been cluster root */
6: $\mathbf{S}_i \leftarrow \{i\}$ /* this nodes sub-tree set */
7: $\mathbf{B} \leftarrow \emptyset$ /* list of bad actors */
8: **if** $i \in \mathbf{C}_{\text{init}}$ **then**
9: $d_{\text{root}} \leftarrow 0$
10: $t \leftarrow 1$
11: $lastUpdated \leftarrow$ `currentTime()`

Algorithm 9 `updateDistanceTree`(M)

1: $\mathbf{B} \leftarrow \mathbf{B} \cup M.\mathbf{B}$ /* synchronize bad actor list */
2: **if** $d_{ji} + M.d_{\text{root}} < d_{\text{root}}$ **then**
/* a shorter path to a cluster root has been found */
3: $parent \leftarrow j$
4: $d_{\text{root}} \leftarrow d_{ji} + M.d_{\text{root}}$
5: **else if** $i = M.parent$ **then**
/* this node i is the parent of the sending node j */
6: $\mathbf{N}_c \leftarrow \mathbf{N}_c \cup \{j\}$
7: $\mathbf{S}_j \leftarrow M.\mathbf{S}$
8: **else if** $i \neq M.parent$ **then**
/* this node i is not parent of the sending node j */
9: $\mathbf{N}_c \leftarrow \mathbf{N}_c \setminus \{j\}$
10: $\mathbf{S}_j \leftarrow M.\mathbf{S}$
11: **if** $parent = j$ **then**
/* sending node j is the parent of this node i */
12: **if** $M.newRootID = i$ **then**
/* this node j is the new cluster root */
13: $parent \leftarrow \emptyset$
14: $t \leftarrow t + 1$
15: $z \leftarrow z + 1$
16: $lastUpdated \leftarrow$ `currentTime()`
17: **else**
18: $d_{\text{root}} \leftarrow d_{ji} + M.d_{\text{root}}$

Algorithm 10 `calculateGoodActors()`

1: **for** $j \in \mathbf{B}$ **do**
2: **if** $\mathbf{S}_j = \{j\}$ **then**
3: $\mathbf{B} \leftarrow \mathbf{B} \setminus \{j\}$
4: **for** $\ell \in \mathbf{N}_c \setminus \{j\}$ **do**
5: `conflictDurationReset`$(\{j\}, \{\ell\})$
6: **return** $\mathbf{B}$

Algorithm 11 `calculateBadActors()`

1: **for** $j \in \mathbf{N}_c \setminus \mathbf{B}$ **do**
2: **for** $\ell \in (\mathbf{N}_c \setminus \mathbf{B}) \setminus j$ **do**
3: **if** $\mathbf{S}_j \cap \mathbf{S}_\ell \neq \emptyset$ **then**
4: **if** `conflictDurationExceeded`(j, ℓ) **then**
5: $\mathbf{B} \leftarrow \mathbf{B} \cup \{j\}$
6: **else**
7: `conflictDurationIncreased`(j, ℓ)
8: **return** $\mathbf{B}$

Algorithm 12 `resetDistanceTree()`

1: $d_{\text{root}} \leftarrow \infty$
2: $parent \leftarrow \emptyset$
3: $\mathbf{N}_c \leftarrow \emptyset$

Algorithm 13 `populateMessageMetaData`$(\tilde{M})$

1: $\tilde{M}.senderID \leftarrow i$
2: $\tilde{M}.t \leftarrow t$
3: $\tilde{M}.parent \leftarrow parent$
4: $\tilde{M}.newRootID = \emptyset$

Algorithm 14 `blockBadActors()`

1: $\mathbf{N}_c \leftarrow \mathbf{N}_c \setminus \mathbf{B}$
2: **if** $parent \in \mathbf{B}$ **then**
3: $parent \leftarrow \emptyset$
4: $d_{\text{root}} \leftarrow \infty$

can potentially be addressed my modifying the functionality of the subroutines `calculateBadActors`() and `calculateGoodActors`().

For brevity, the pseudocode for Distributed Swarm k-Means (without IDRS) is combined with that for Resilient Swarm k-Means (with IDRS) in Algorithms 7-14. Color is used to indicate which lines belong to which variants. The set of initial cluster centers $\mathbf{C}_{\text{init}}$ is provided (by a user or some other means such as random selection) and each robot knows its own ID number i and the value of $z_{\max}$. An additional user-defined parameter $T_{\min}$ is compared with the time since the root shifted $lastUpdated$ to prevent root transfer from occurring too quickly. Each robot tracks the size of its sub-tree s_i and the sub-tree sizes of each neighbor j using an array entry s_j indexed by j.

The shortest spanning forest distance data is updated to reflect most current information from all neighbors using the subroutine `updateDistanceTree`(), which performs two important tasks. First, it runs the distributed Dijkstra's reverse search that provides each node with its distance to the closest root. Second, it detects if/when this robot has been promoted to be a root—in which case this robot increases the value of t and restarts the distributed distance calculation.

The algorithm works in a distributed fashion that relies on local message passing. Each distributed iteration (at the swarm level) is associated with a unique number t. Messages from previous iterations are ignored, as are messages from beyond the allowed communication radius d (lines 5-6). Whenever the root increases the iteration number, then the shortest path tree distance calculation is reset (Lines 7-10). If the IDRS is used, then the sub-tree set is updated (line 15). Basic message data (iteration, tree distances, etc.) is populated using the subroutine `populateMessageMetaData`(). If this robot is the root ($d_{\text{root}} = 0$), has not exhausted its root passing threshold ($z \leq z_{\max}$), and the tree has stabilized for some a user-defined amount of time (`currentTime`()$-lastUpdated \geq T_{\min}$), then this robot chooses the new cluster root (its child with the most descendants), and adds this data to the outgoing message (lines 19-21).

Lines involved with the IDRS appears blue. If the IDRS is used, then messages from suspected malicious robots are also ignored (line 5). Avoiding problems caused by start-up effects (when the minimum spanning forest has not yet converged) is accomplished by defining a "conflict duration", and then only adding robots to the bad actor list if they report conflicting data for longer than the conflict duration. Conflicts are defined as an ordered tuple. The conflict (j, ℓ) means that j is suspected of being malicious due to a conflict with ℓ. The duration for which one node has conflicted with another is incremented inside the subroutine `calculateBadActors`() (line 12). The conflict duration of all non-bad-actors are also reset whenever there is a change in the root node (line 9, not shown for brevity) using the subroutine `resetConflictDurations`$(\mathbf{N}_c \setminus \mathbf{B}, \mathbf{N}_c)$, which resets all timers for conflicts (j, ℓ) such that $j \in \mathbf{N}_c \setminus \mathbf{B}$ and $\ell \in \mathbf{N}_c \setminus j$.

Robots are removed from the bad actor list if the sub-tree set they report contains only themselves inside the subroutine `calculateGoodActors`() (line 13). Finally, robots suspected of being malicious are not allowed to participate as parents or children in the shortest path tree calculation (line 14).

6 Experiments

We perform a series of experiments in both simulation and in hardware testbeds using a homogeneous swarm of 25 Kilobot robots, pictured in Figure 1. Kilobots are identical, low-cost robots, 33mm in diameter, controlled by an AtMega328 microprocessor [15]. They use infrared LEDs and sensors to communicate with other Kilobots within a 100mm radius, and convey information though an RGB LED. The simulations use a real-time Kilobot simulator written in C.

6.1 Silhouette Coefficient Performance Metric

We use the silhouette coefficient to evaluate clustering performance [14]. This statistic requires, for all n nodes, the average distance to all nodes within the cluster, a_i, and the average distance to all nodes not in the cluster, b_i. For each node, a silhouette value is calculated $c_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$. This value ranges $-1 \leq c_i \leq 1$, such that negative values indicate bad clustering and positive values indicate good clustering. The final silhouette coefficient, summarizing the clustering accuracy for all nodes, is the maximum of the n silhouette values. If the whole swarm is classified as a single cluster, the silhouette coefficient is zero.

6.2 Experimental Process

We investigate six different cases: the Distributed (non-IDRS) variants of Swarm k-Means and Swarm DBSCAN are both tested with and without malicious attacks. The Resilient (IDRS) variants are tested with malicious attacks. Each of the six cases is evaluated using repeated trials in both simulation and hardware.

Repeated trials involved 15 unique cluster configurations. Each configuration is randomly generated as follows: three 2D coordinates serve as cluster centers in a generative model, 25 points are generated by sampling three Gaussian distributions, each with a mean located at one of the centers. An *oracle silhouette coefficient* is calculated using the perfect knowledge of which position corresponds to which Gaussian. This oracle value defines the silhouette coefficient of "perfect" performance achievable by an oracle with complete knowledge of sample generation. While any real algorithm should not be expected to outperform an oracle, the oracle silhouette coefficient provides useful context for evaluating the relative performance of the six cases.

On the real-time simulator, we ran six algorithm: the three k-Means Algorithms and the three DBSCAN Algorithms. Each algorithm ran on the 15 configurations three time each, for a total of 45 trials per algorithm (270 trials total). We ran the Attacked k-Means, the Attacked DBSCAN, the Resilient k-Means, and the Resilient DBSCAN with one, two, and three adversaries split evenly among the 45 trials per algorithm. During each trial, the adversaries (and for k-Means the k roots) were chosen uniformly at random from members of the swarm. The resulting starting configuration was used once for each of the six cases. Hardware testbed experiments were performed in the same manner, running 15 trials for the three k-Means scenarios, and 30 trials for each of three DBSCAN scenarios. Overall, we performed 135 hardware testbed tests and 270 tests in the real-time simulator.

Table 1. Silhouette Coefficient as Percentage of Oracle on Simulator

	Distributed	Attacked	Resilient
k-Means (45 trials per algorithm)	85%	8%	74%
DBSCAN (45 trials per algorithm)	97%	66%	95%

Table 2. Silhouette Coefficient as Percentage of Oracle on Hardware

	Distributed	Attacked	Resilient
k-Means (15 trials per algorithm)	94%	40%	71%
DBSCAN (30 trials per algorithm)	98%	72%	99%

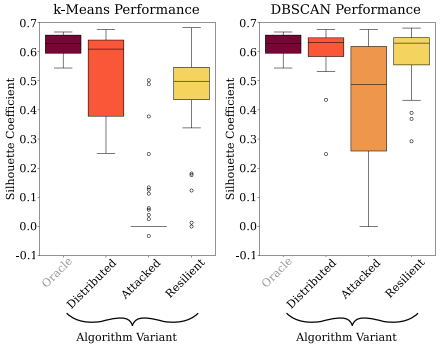

Fig. 5. Comparative performance of the k-Means and DBSCAN algorithms on the simulator (45 trials per algorithm).

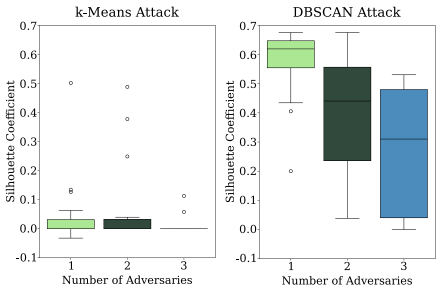

Fig. 6. The comparative effect of the number of adversaries on the simulator (15 trials per number of adversaries).

7 Discussion of Results

We find that Distributed Swarm DBSCAN tends to produce more accurate clustering than the Distributed Swarm k-Means Algorithm. On average, Distributed Swarm k-Means performed at, on average, 85% of the "perfect" oracle silhouette coefficient, while Distributed Swarm DBSCAN performing, on average, at 97% of the oracle (Table 1). Similar trends appear in simulation and hardware experiments (Tables 1 and 2).

Hypothesis tests show a statistically significant loss in performance in the presence of a malicious robot ($p < 1 \times 10^{-6}$ for the simulated swarm using a one-tailed paired t-test; $p < 2.6 \times 10^{-5}$ for the hardware swarm using a nonparametric Kolmogorov-Smirnov (KS) test). The k-Means attack resulted in an average silhouette coefficient near zero, performing at 8% of the oracle. By comparison, the DBSCAN algorithm was less susceptible to attacks, performing 66% of the oracle in the presence of one or more adversaries (Table 1). This implies that even a single malicious robot can have disastrous consequences on a swarm's emergent behavior. Swarm k-Means, with its top-down tree structure, appears more vulnerable to disruption than the bottom-up DBSCAN. This reinforces the idea that potential for disruption by a malicious robot increases with centralization.

Using the IDRS produced a statistically significant improvement in accuracy for both the Swarm k-Means and Swarm DBSCAN Algorithms ($p < 1.0 \times 10^{-5}$ for the simulated swarm using a one-tailed paired t-test; $p < 0.04$ for the hardware swarm using a nonparametric KS test). On average on the simulator, the Resilient k-Means restored performance to within 10% of its non-attacked variant and Resilient DBSCAN restored performance to within 3% of its non-attacked variant. For the k-Means algorithm, though performance improved with the IDRS, it did not return to pre-attack levels, with a statistically significant difference when compared with the non-attacked Distributed variant ($p < .002$). The

malicious robot caused more disruption to the algorithm than the IDRS could undo within the time taken to isolate the attack. The performance of Resilient DBSCAN, however, restored clustering accuracy to near-original levels, with no statistically significant difference between performance for the Distributed and Resilient variants. In total, these results suggest that a well-programmed distributed IDRS can protect swarms in hostile situations, partially or fully.

Figure 6 contrasts how, depending on the algorithm, a greater number of adversaries may or may not worsen the algorithm's performance. For the k-Means attack, varying the number of adversaries in the swarm has very little relationship with the attacks ability to disrupt clustering. A single malicious robot has the potential to hijack all three roots, and so adding more adversarial robots did not increase its potential for disruption. In contrast, each additional malicious robot present during DBSCAN Algorithm's caused a notable drop in accuracy and a greater variation in performance.

8 Conclusion

In this paper, we investigate how malicious masquerade attacks can affect two distributed swarm clustering algorithms—one based on the k-Means algorithm and another based on the DBSCAN algorithms. We analyze how the algorithms' resilience to such attacks can be increased through the use of a distributed Intrusion Detection and Response System (IDRS). These experiments were performed on both a hardware and simulated testbed of swarm robots. We find that the Distributed Swarm k-Means Algorithm is more susceptible to attack the Distributed Swarm DBSCAN Algorithm. Results also show that increasing the number of adversarial robots caused larger disturbances in the DBSCAN Attack, while more adversaries had less effect on the k-Means Attack. Finally, we find that distributed IDRS can largely restore the swarm's performance.

Acknowledgments This work was partially supported by a joint Northrop Grumman and UMD Seedling Grant and by ONR grant N000142012712.

References

1. Anikin, I.V., Gazimov, R.M.: Privacy preserving DBSCAN clustering algorithm for vertically partitioned data in distributed systems. In: 2017 International Siberian Conference on Control and Communications (SIBCON). pp. 1–4. IEEE (2017)
2. Ben Salem, M.: Towards effective masquerade attack detection (2012), https://academiccommons.columbia.edu/doi/10.7916/D8J96DBT
3. Dasgupta, S., Frost, N., Moshkovitz, M., Rashtchian, C.: Explainable k-means and k-medians clustering. In: Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria. pp. 12–18 (2020)
4. Gil, S., Kumar, S., Mazumder, M., Katabi, D., Rus, D.: Guaranteeing spoof-resilient multi-robot networks. Autonomous Robots 41 (08 2017)
5. Götz, M., Bodenstein, C., Riedel, M.: HPDBSCAN: highly parallel DBSCAN (2015)

6. Higgins, F., Tomlinson, A., Martin, K.: Threats to the swarm: Security considerations for swarm robotics. International Journal on Advances in Security 2 (2009)
7. Kolias, C., Kambourakis, G., Maragoudakis, M.: Swarm intelligence in intrusion detection: A survey. Computers & Security 30(8), 625 – 642 (2011), http://www.sciencedirect.com/science/article/pii/S016740481100109X
8. Laing, T., Martin, K., Ng, S., Tomlinson, A.: Security in Swarm Robotics, pp. 42–66. IGI Global (Dec 2015)
9. Ling, R.F.: On the theory and construction of k-clusters. The Computer Journal 15(4), 326–332 (01 1972), https://doi.org/10.1093/comjnl/15.4.326
10. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability. vol. 1, pp. 281–297. Oakland, CA, USA (1967)
11. Mao, Y., Xu, Z., Li, X., Ping, P.: An optimal distributed k-means clustering algorithm based on cloudstack. In: 2015 IEEE International Conference on Information and Automation. pp. 3149–3156. IEEE (2015)
12. McCune, R., Madey, G.: Decentralized k-means clustering with MANET swarms. In: Proceedings of the 2014 Symposium on Agent Directed Simulation. pp. 1–8 (2014)
13. Patel, S., Patel, V., Jinwala, D.: Privacy preserving distributed k-means clustering in malicious model using zero knowledge proof. In: International Conference on Distributed Computing and Internet Technology. pp. 420–431. Springer (2013)
14. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20, 53–65 (1987)
15. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science 345(6198), 795–799 (2014), https://science.sciencemag.org/content/345/6198/795
16. Schranz, M., Umlauft, M., Sende, M., Elmenreich, W.: Swarm robotic behaviors and current applications. Frontiers in Robotics and AI 7, 36 (2020)
17. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response system. International Journal of Information and Computer Security 1 (01 2007)
18. Whelan, C., Harrell, G., Wang, J.: Understanding the k-medians problem. In: Proceedings of the International Conference on Scientific Computing (CSC). p. 219. The Steering Committee of The World Congress in Computer Science, Computer (2015)
19. Xu, D., Tian, Y.: A comprehensive survey of clustering algorithms. Annals of Data Science 2 (08 2015)
20. Yang, K., Gao, Y., Ma, R., Chen, L., Wu, S., Chen, G.: DBSCAN-MS: Distributed density-based clustering in metric spaces. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 1346–1357. IEEE (2019)
21. Zhou, J., Zhang, Y., Jiang, Y., Chen, C.P., Chen, L.: A distributed k-means clustering algorithm in wireless sensor networks. In: 2015 International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS). pp. 26–30. IEEE (2015)