Received 9 February 2024, accepted 4 March 2024, date of publication 7 March 2024, date of current version 18 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3374409

RESEARCH ARTICLE

ACapDS: An Energy-Efficient and Fault-Tolerant Distributed Capacitated Dominating Set Algorithm for Industrial IoT

OZKAN ARAPOGLU[1], UMUT CAN CABUK[1,2], (Member, IEEE), ORHAN DAGDEVIREN[3], (Member, IEEE), AND MOHARRAM CHALLENGER[4,5], (Member, IEEE)

[1]International Computer Institute, Ege University, 35040 İzmir, Turkey
[2]Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182, USA
[3]Department of Computer Engineering, Ege University, 35040 İzmir, Turkey
[4]Department of Computer Science, University of Antwerp, 2020 Antwerp, Belgium
[5]Flanders Make Strategic Research Center, 3001 Leuven, Belgium

Corresponding author: Umut Can Cabuk (ucabuk@sdsu.edu)

This work was supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under Grant 215E115.

ABSTRACT Design and analysis of energy-efficient and fault-tolerant dominating set (DS) algorithms are vital tasks for Industrial Internet of Things (IIoT) scenarios, as operational efficiency is a key objective in most industries. Such algorithms must maintain topology control, clustering, routing, and data aggregation. An IIoT deployment is said to pose the self-stabilization property if it can always recover to a steady state in a bounded time without any interruption whenever it is started at an unstable state. Self-stabilization is a fruitful principle for building fault-tolerant IIoT deployments. This paper proposes a novel distributed fault-tolerant capacitated DS (capDS) algorithm for IIoT systems. The algorithm is the first self-stabilizing capDS approach to the best of our knowledge. Proofs concerning the asynchronous behaviors of the algorithm, as well as the self-stabilization feature with regard to convergence and closure properties, are provided. Besides, our theoretical analysis showed that the algorithm's approximation ratio is 6 for IIoT setups implemented as unit disk graphs. Measurements made using our testbed of 40 IRIS motes and the extensive TOSSIM simulations revealed that the proposed algorithm is up to 55% better in terms of coefficient of variation, requires up to 61% fewer moves, causes up to 13% less data traffic, and consumes up to 14% less energy when compared to a randomized approach and a minimum ID priority-based approach.

INDEX TERMS Industrial Internet of Things (IIoT), minimum capacitated dominating set, distributed algorithms, self-stabilizing algorithms, fault tolerance.

I. INTRODUCTION

The Internet of Things (IoT) is a revolutionary paradigm that allows interconnecting a vast amount of devices to each other (and to the internet), and thus, it enables continuously gathering information from the environment and supports making collaborative decisions [1]. An industrial IoT (IIoT) system consists of independent and interconnected computing nodes that are capable of sensing data from devices, products, buildings, and the environment [2], [3], [4], [5]. These nodes also help monitor and manage physical resources like electrical energy or waste.

Wireless technologies have become pervasive in IIoT due to their irreplaceable benefits, including wider area coverage, mobility, higher scalability as well as lower maintenance costs [6], [7], [8], [9]. As hundreds, even thousands, of geographically scattered nodes may form these networks; therefore, the design, analysis, and implementation of distributed algorithms targeting to solve various IIoT problems are vital in the field of research [10]. Depending on

The associate editor coordinating the review of this manuscript and approving it for publication was Stefano Scanzio.

the application setup, IIoT nodes may be deployed in harsh environments including but not limited to ocean beds, tunnels, and underground mines where nodes are left unguarded and prone to failures. Hence, fault tolerance is of the utmost importance for IIoT. This is where self-stabilization comes in handy, as it is a feasible technique for introducing fault tolerance to IIoT applications. An IIoT instance is said to be self-stabilizing [11] if it can begin from any state and regain a steady-state (also called a valid or legitimate state) in a limited time without any interrupt.

A self-stabilizing distributed algorithm should have the closure and convergence properties [12]. In an IIoT deployment, each node may (or may not) execute the same instructions and may modify its state concurrently, provided that the instructions executed by each node check the preconditions of its rules. If all the preconditions of a rule are satisfied, then the rule can be enabled. When a node has at least one enabled rule, it is called "privileged," and it can be selected by a scheduler to execute the given action for the particular node. If a privileged node changes its state, it is called a "move". It is a common practice to assume that each move is executed atomically. An unfair distributed scheduler may choose a nonempty subset of the privileged nodes for execution (i.e., a privileged node may execute its rules in a fully asynchronous way.), and such a scheduler can be highly suitable and is realistic for the IIoT case [13].

A. PROBLEM STATEMENT

An IIoT deployment can safely be modeled as a unit disk graph (UDG) $G = (V,E)$, where V is the set of included nodes (i.e., vertices) and E is the set of links (i.e., edges) connecting the nodes [14], [15]. Any arbitrary node pair, say v_1 and v_2, are considered neighbors and the edge (v_1,v_2) exists in E if and only if the Euclidean distance between nodes v_1 and v_2 is shorter than a predefined threshold (i.e., transmission range). Some domination problems of Graph theory (and their solutions), including but not limited to the maximal independent set (MIS), dominating set (DS), and connected dominating set (CDS) paradigms, can be exploited to enhance the energy efficiency [16] and fault tolerance [17] in distributed computing systems, like IIoT deployments.

An independent set of a graph $G(V, E))$ is a subset IS of V, such that there are no two (or more) adjacent (i.e., neighboring) nodes in IS. The node cardinality of an independent set implies its size. After an arbitrary search process, if an independent set can no longer be extended with other nodes in G, then this set is called a maximal independent set. Further, the largest of such independent sets is called the maximum independent set. Nonetheless, a dominating set [14], [18], consisting of nodes D, is another subset of V, where every node included is either a member of D or is a neighbor of a member of D. Formally, $D = \{v \in V{:}\forall t \in V - D{:}\ \exists d \in D{:}(d, t) \in E\ \}$. Likewise, CDS is a special (i.e., connected) case of DS.

A capacitated DS (CapDS) S is a subgraph of G (also a subset of V), where S is a DS, and every single non-dominator (also called a dominatee) is associated with a dominator, provided that the number of dominatees a dominator can be matched with may not be larger than its (preset) capacity (c). Let a dominatee be mapped to a dominator with the mapping function: $V/S \rightarrow S$. In a more formal notation, CapDS can be expressed as $S = \{v \in D{:}\ |\{u \in V - D : m(u) = v\}| \leq c\ \}$. Finding efficient ways to minimize the set S is an open problem and is NP-hard [19]. The design of a CapDS algorithm is crucial for achieving low-energy clustering and convergent load-balancing methods for IIoT deployments, where cluster sizes are bounded by a cap. We can more formally define our problem using the following formulations. The decision variables are:

$$x_i = 1 \iff i \in \text{CDS}$$
$$z_{ij} = 1 \iff i \in \text{CDS, and let j be assigned to i}$$

The objective function is:

$$min \sum_{i \in V} x_i$$

And the limitations are:

$$\sum_{j \in \Gamma(i) \cup \{i\}} x_j \geq 1 \qquad \forall i \in V$$
$$z_{ij} \leq x_i + x_j \qquad \forall ij \in E,\ i \in V$$
$$z_{ij} \leq 2 - x_i + x_j \qquad \forall ij \in E,\ i \in V$$
$$\sum_{j \in \Gamma(i)} z_{ij} \leq x_i c_i + (1 - x_i) \qquad \forall i \in V$$
$$x_i, z_{ij} \in \{0, 1\} \qquad \forall i, j \in E,\ \forall i \in V$$

where x_i shows if node i dominates or not, and $\Gamma(i)$ is neighbors list of node i.

B. CONTRIBUTIONS

This paper studies the CapDS problem, particularly for the IIoT case. Our contribution to the field can be summed up under three directions and are listed below:

- Although finding dominating sets of a graph has been well studied since the graph theory emerged [17], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], as far as we know, a self-stabilizing distributed algorithm for the CapDS problem was not developed. Hereby, we proposed the first capacitated MIS and capacitated CDS algorithms in earlier works [31], [32]. Note that any maximal independent set is also a dominating set. However, not vice versa. A dominating set may include two neighboring nodes in the DS, but this is not allowed for the MIS case. So, this paper introduces the first self-stabilizing distributed CapDS algorithm, and it is tailored for IIoT.
- We present a theoretical analysis of the algorithm by evaluating various aspects. We also proved the correctness of our algorithm per the closure and convergence analyses. For the widely used UDG model, it is proven that the approximation ratio of our algorithm is 6.

We study the upper limit of the move count produced by the proposed algorithm to estimate its worst-case resource occupation.

- We implemented the algorithm on testbeds made of IRIS sensor motes and the TOSSIM simulator environment against different parameters, such as the node count and average node degrees. From extensive measurements, we show that the proposed algorithm is favorable in terms of move counts, received byte counts, coefficient of variation, energy consumption, and the time to first node failure.

Consequently, the related works are proposed in Section II. The proposed algorithm is represented in Section III. The theoretical analysis is given in Section IV. Section V includes our testbed experiments and simulations. The conclusions are finally given in Section VI.

II. RELATED WORKS

DS has been studied pervasively by lots of researchers such that it is used for efficient identification of web communities [33], food webs [34], a quantitative analysis of secondary RNA structure in bioinformatics [35] and distributed computing [36], [37]. We can mainly categorize the algorithms as central and distributed. Due to time, energy, and communication constraints, central algorithms are unsuitable for distributed systems. Moreover, managing large-scale networks from a central node is generally inefficient under various dynamic network scenarios. DS algorithms are widely used for clustering and routing. Thus, plenty of efficient algorithms for various DS problems and their derivations, such as connected DS, are proposed in the literature as central [38] and distributed [29]. Since all algorithms mentioned in this paragraph are not self-stabilizing, they are out of our scope.

A self-stabilizing algorithm can cope with failures to provide fault tolerance in IIoT. Under a central scheduler by using a dominating bipartition concept, Hedetniemi et al. [39] presented two self-stabilizing DS algorithms. Xu et al. [40] presented a synchronous self-stabilizing approach calculating a minimal DS. The algorithm has O(n) rounds time complexity where n is the node count. Goddard et al. designed a self-stabilizing algorithm running by a centralized scheduler to find a minimal DS. Turau [13] proposed a distributed self-stabilizing DS algorithm using a distributed scheduler. Its time complexity is O(n). Another self-stabilizing DS algorithm that stabilizes in linear time under a distributed scheduler is Goddard et al.'s [41]. Chiu et al. [42] proposed a self-stabilizing DS approach having $4n$ move and running on an unfair distributed scheduler. Although all these algorithms are fault-tolerant, they do not provide a capacity constraint.

In 2007, the (soft) capacitated domination problem was put forth by Kao and Liao [43] as a matter of finding a DS with the minimum node cardinality that considers the capacity and demand parameters. They formulated a linear-time 3/2-approximation algorithm. From a different standpoint, Dom et al. [44] elaborated on the characteristics of the capacitated DS considering the parameterized complexity perspective, showing that CapDS is W[1]-hard if parameterized by the treewidth and the solution size k of the CapDS.

The first bi-dimensional variation of the CapDS problem, namely the planar CapDS problem, was pointed out by Dom et al. [44] as an open research problem. It was later resolved by Bodlaender et al., in [45], where they revealed that the planar CapDS is W[1]-hard. For the minimum CapDS problem, the first distributed solution was proposed by Kuhn and Moscibroda [46], as earlier works preferred centralized approaches. Kao and Chen [47] presented algorithms provided that the tree width and the maximum capacity of the nodes were utilized as parameters. Another approximation approach for the minimum CapDS problem was proposed by Shang and Wang in [48]. This was a centralized algorithm and is a good starting point in the journey of unveiling the CapDS problem. However, exact solutions are also not unreachable. Such an approach that solves the CapDS problem in $O(1.89^n)$ time-complexity class was proposed by Cygan et al. [49]. Potluri and Singh [50] showed that heuristics could also be useful. They proposed a heuristic approach (with several editions) to construct CapDS. In addition, it provides better performance when compared to competitors.

Kao et al. [51] surveyed the capacitated domination problem's complexity and the existing approximation approaches. Liedloff et al. [52] presented a solution for the CapDS problem in $O^*(1.8463^n)$ time by benefiting from dynamic programming over subsets. Further, Becker [53] suggested a polynomial-time approximation method for CapDS, particularly for planar and unweighted graphs, where the maximum capacity and maximum demand are both bounded. On the other hand, Li et al. [54] proposed a local search algorithm to solve CapDS. Our extensive literature review yielded no self-stabilizing algorithm for the CapDS problem. The algorithm we proposed is the first successful attempt in this sense.

III. PROPOSED ALGORITHMS

How the proposed method works is formally expressed in Algorithm 1 (referred to as A_{CapDS}). Yet all the acronyms and definitions of A_{CapDS} are shown in Table 1. A unique identifier, denoted by id_i, is assigned to each node i in the IIoT deployment. N_i and c_i denote the neighbors of the node i and its capacity value, respectively. S_i is the local state of an arbitrary node i, which may be either one of the "OUT" or "IN" states. For any node i, having S_i = OUT shows that it is already not an element of CapDS. Otherwise, if it is in CapDS, S_i must be "IN". $Dominator_i$ stands for the node i's dominator. The $NULL$ is represented by $\perp$. The term $Dominatees_i$ is the set of dominatees of the node i. Whenever node i's capacity becomes (resp. is not) full, and i is (resp. is not) the dominator node of all dominatees in $Dominatees_i$, then $isFull_i$ set to true (resp. false).

Algorithm 1 A_{CapDS}

```
Inputs. $id_i$, $N_i$, $c_i$.
Variables. $S_i$, $Dominator_i$, $Dominatees_i$, $IsFull_i$
Macros.
$EmptyCapacity_i$: $|c_i - |Dominatees_i||$.
$CanDominatees_i$: $\{j \in N_i | S_j = OUT \wedge Dominator_j = \perp \wedge j \notin Dominatees_i\}$.
$CanDominators_i$: $\{j \in N_i | S_j = IN \wedge i \in Dominatees_j \wedge EmptyCapacity_j \geq 0\}$.
$MinNbr_i$: $min\{j \in N_i\}$, $MaxNbr_i$: $max\{j \in N_i\}$.
$MaxEmptyNbr_i$: $j \in N_i | S_j = IN \wedge \forall t \in N_i (S_t = IN \wedge j \neq t \wedge EmptyCapacity_j \geq EmptyCapacity_t)$.
$IsFullMac_i \in \{true, false\}$:
  if($S_i = IN \wedge EmptyCapacity_i = 0 \wedge \forall j \in Dominatees_i[Dominator_j = i]) then IsFullMac_i := true$.
  if($S_i = OUT \vee (S_i = IN \wedge (EmptyCapacity_i \neq 0 \vee \exists j \in Dominatees_i[S_j = IN \vee Dominator_j \neq i]))) then IsFullMac_i := false$.
Rules.
Rule 1. if $S_i = IN \wedge EmptyCapacity_i \neq 0 \wedge \exists j \in N_i[\, S_j = IN \wedge j < i \wedge \neg IsFull_j]$ then
  $S_i := OUT,\ Dominator_i := \perp$
Rule 2. if $S_i = IN \wedge |Dominatees_i| > c_i$ then
  repeat
    Pick $MaxNbr_i \in Dominatees_i$
    $Dominatees_i := Dominatees_i \backslash \{MaxNbr_i\}$
  until $S_i \neq IN \vee |Dominatees_i| <= c_i$
  $IsFull_i := IsFullMac_i$
Rule 3. if $S_i = IN \wedge EmptyCapacity_i > 0 \wedge CanDominatees_i \neq \emptyset$ then
  repeat
    Pick $MinNbr_i \in CanDominatees_i$
    $Dominatees_i := Dominatees_i \cup \{MinNbr_i\}$
    $CanDominatees_i := CanDominatees_i \setminus \{MinNbr_i\}$
  until $S_i \neq IN \vee EmptyCapacity_i = 0 \vee CanDominatees_i = \emptyset$
  $IsFull_i := IsFullMac_i$
Rule 4. if $S_i = IN \wedge \exists j \in Dominatees_i[\, (Dominator_j \neq i \wedge Dominator_j \neq \perp) \vee j \notin N_i \vee S_j = IN]$ then
  repeat
    $Dominatees_i := Dominatees_i \backslash \{j\}$
  until $S_i \neq IN \vee \forall j \in Dominatees_i[\, (Dominator_j = i \vee Dominator_j = \perp) \wedge j \in N_i \wedge S_j \neq IN]$
  $IsFull_i := IsFullMac_i$
Rule 5. if $S_i = OUT \wedge Dominator_i = \perp \wedge CanDominators_i \neq \emptyset$ then
  Pick $MaxEmptyNbr_i$ from $CanDominators_i$
  $Dominator_i := MaxEmptyNbr_i$
Rule 6. if $S_i = OUT \wedge Dominator_i \neq \perp \wedge [i \notin Dominatees_{Dominator_i} \vee Dominator_i \notin N_i \vee S_{Dominator_i} = OUT]$ then
  $Dominator_i := \perp$
Rule 7. if $S_i = OUT \wedge Dominator_i = \perp \wedge \forall j \in N_i[\, (S_j = OUT \wedge (i < j \vee Dominator_j \neq \perp)) \vee (S_j = IN \wedge IsFull_j)]$ then
  $S_i := IN,\ Dominatees_i := \emptyset,\ IsFull_i := IsFullMac_i$
Rule 8. if $S_i = IN \wedge \neg R1 - R7 \wedge IsFull_i \neq IsFullMac_i$ then
  $IsFull_i := IsFullMac_i$
```

EmptyCapacity$_i$ is used to keep track of the highest number of extra dominatee assignments to the dominator i, required to complete its preset capacity. *CanDominatees*$_i$ represents the set of dominatee candidates which can be associated with the dominator node i. Likewise, *CanDominators*$_i$ is the set of possible dominator nodes, one of which can be selected as the dominator node for the dominatee i. Parameters *MinNbr*$_i$ and *MaxNbr*$_i$ represent the maximum and minimum neighbors of node i, subsequently. Lastly, *IsFullMac*$_i$ is designated to set the value of the *IsFull*$_i$ parameter.

The proposed algorithm has eight defined rules (each referred to as R#). The nodes in the OUT state (i.e., the dominatees) execute R5, R6, and R7, whereas IN nodes (i.e., the dominators) execute the others. Over and above, R8, R7, and R1 are directly related to building a DS, while the remaining define and enforce the capacity constraint. R1 implies that if a node i is having IN state, it has an IN neighbor node which has a lower ID, its capacity is not full and also *isFull* = false, then the node i transforms into the OUT state and sets its *Dominator*$_i$ to *NULL*. Any two IN nodes that do not have their capacities full are not eligible to be neighbors, yet this property is maintained by R1. When a node i has IN state, and the cardinality of the dominatees of the node i is greater than the capacity, then it runs R2 as well as deletes the nodes from its dominatees until the cardinality of this set becomes not greater than the capacity The cardinality of *Dominatees*$_i$ can overflow its capacity in the start state. R2 exactly addresses this issue. Per R3, the dominators select their dominatees, considering their available capacities. If the cardinality of the set of dominatees of an IN node i is smaller than its capacity and there exists one or more candidate dominatee that did not choose a dominator yet, it executes R3 and writes the dominatee candidates into its dominatees set until its capacity becomes full. R4 dictates that if a node j is maintained in the dominatees set of the dominator node i, but the node j is associated with another dominator, then the node j is removed from the node i's dominatees set to prevent a double-affiliation issue.

If any node i is in the OUT state, there is one or more IN neighbors of i in its dominatees set, and the dominator of i is *NULL*; then the node i runs R5 by selecting a dominator node among the ones in its *CanDominators* set and assigning it as its dominator. This results in a dominator request and is replied to by a dominatee that does not currently have an agreed dominator and is already selected by a dominator. Hence, that node achieves a matching as a result of R5. A dominatee node pointing to an incorrect dominator resets its dominator value by executing R6.

In case of a bad match-up between a dominatee and a dominator, if any, R4 and R6 together help address this issue. As R7 dictates, if a node i is in the OUT state, it is not associated with a dominator node, each dominatee neighbor of i with a lower ID is associated with a dominator, and all dominator neighbors of the node i have its capacity full, then the node i self-assigns itself a dominator by altering its state to IN. Yet, R7 assures that there is a dominator node in every cluster, excluding the fully occupied dominators and their dominatees. Nevertheless, *IsFull*$_i$ allows a dominator and its neighbors to communicate and inform each other regarding whether the dominator's capacity is full or not. If none of the rules R1 to R7 are enabled, and the variable *IsFull*$_i$ is false, then the node i can update it by eventually executing R8.

TABLE 1. Symbols and abbreviations.

Acronym	Description
G	An undirected graph.
V	Set of vertices of G.
E	Set of edges of G.
i	Arbitrary node in V.
id_i	Identifier of node i.
c_i	Capacity of node i.
N_i	Neighbors of node i.
S_i	Node i's state, $S_i \in \{OUT,\ IN\}$.
$\perp$	Null value.
$Dominator_i$	Dominator of i.
$Dominatees_i$	Dominatees set of node i.
$EmptyCapacity_i$	Number of empty space of $Dominatees_i$ of node i.
$CanDominators_i$	Candidate dominators of node i.
$CanDominatees_i$	Candidate dominatees of node i.
$MinNbr_i$	The neighbor of node i with minimum ID.
$MaxEmptyNbr_i$	The neighbor of node i with maximum $EmptyCapacity$.
$IsFull_i$	Each dominatee in $Dominatees_i$ points to i as dominator and $EmptyCapacty_i$ is full, $IsFull_i \in \{True,\ False\}$.

TABLE 2. The steps of A_{CapDS} when it converges as in Fig. 1.

	Initial State	1st Step	2nd Step	3rd Step	4th Step
Node 1	$S_1 = IN$ $Dominatees_1 = \{2, 3, 8\}$ $IsFull_1 = False$	R2 $Dominatees_1 = \{2, 3\}$	R8 $IsFull_1 = True$		
Node 2	$S_2 = OUT$ $Dominator_2 = \perp$	R5 $Dominator_2 = 1$			
Node 3	$S_3 = OUT$ $Dominator_3 = \perp$	R5 $Dominator_3 = 1$			
Node 4	$S_4 = OUT$ $Dominator_4 = \perp$	R7 $S_4 = IN$ $Dominatees_4 = \{\}$ $IsFull_4 = False$	R3 $Dominatees_4 = \{5, 6\}$		R8 $IsFull_4 = True$
Node 5	$S_5 = IN$ $Dominatees_5 = \{1, 2\}$ $IsFull_5 = True$	R1 $S_5 = OUT$ $Dominator_5 = \perp$		R5 $Dominator_5 = 4$	
Node 6	$S_6 = OUT$ $Dominator_6 = 3$	R6 $Dominator_6 = \perp$		R5 $Dominator_6 = 4$	

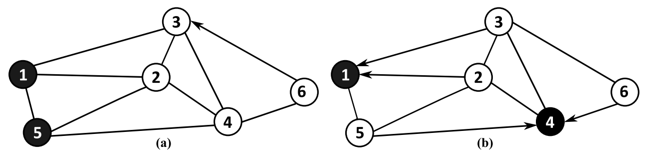

Fig. 1. A sample execution of A_{CapDS} a) Initial state b) Steady state.

Fig. 1 provides an example run of A_{CapDS}, where the black balls represent the nodes that are in the IN state, whereas the white ones represent the nodes in the OUT state, while the capacity is set to two. If a dominatee selects its dominator, it points to its dominator with an arrow. Fig. 1.a. presents the system's initial state. The nodes, excluding 1 and 5, are initially in the OUT state. Nodes 1 and 5 are in IN state, where node 1 is overflowed. Nodes 2, 3, and 4 do not have a dominator, and the variable $IsFull_5$ is false. In the first step, node 1 executes R2, trying to reduce its overflowed capacity. Nodes 2 and 3 execute R5 to set node 1 as their dominator. Node 4 executes R7 since $IsFull_5$ is true. Node 5 executes R1 due to having an IN neighbor (i.e., node 1) with a lower ID. Further, node 6 executes R6 to fix its dominator as *NULL*. In the second step, node 1 executes R8 and sets the $IsFull_1$ variable. Consequently, nodes 1, 2, and 3 are now stabilized. Node 4 executes R3 and adds nodes 5 and 6 into its $Dominatees_4$ set. In the upcoming step, nodes 5 and 6 set node 4 as their dominator. Finally, node 4 executes R8 and sets the $IsFull_4$ variable. Following the completion of these steps, there are no more enabled rules. Fig. 1.b demonstrates the steady state of the given system. Nodes 1 and 4 are in CapDS, and their "isFull" variables are set to true. Node 1 is the dominator of nodes 2 and 3, whereas node 4 dominates nodes 5 and 6. The step-wise convergence of A_{CapDS} can be observed in Table 2, which also includes the executed rules and the corresponding state transitions.

IV. THEORETICAL ANALYSIS

A. CLOSURE

Lemma 1: $Dominator_i = j$ if and only if $i \in Dominatees_j$, provided that a system is in the steady state.

Proof: Assume that the initial system is in a steady state. Additionally, $Dominator_i = j$ but $i \notin Dominatees_j$ by contradiction. As a result of this situation, vertex i runs R6. When $i \in Dominatees_j$ and $Dominator_i \neq j$, it gives rise to two cases. In the first case, if $Dominator_i = \perp$, vertex i executes R5. When it comes to the second case, if $Dominator_i \neq \perp$ and $Dominator_i \neq j$, vertex j executes R4. It is obvious that there is one or more move(s) in the steady state. Therefore, a contradiction is encountered. ■

Theorem 1: The set S closures a CapDS when no vertex is enabled.

Proof: Assume that the state of the system is steady and there is not any enabled vertex. However, S is not a CapDS. There are two cases: (i) S is not a DS, or (ii) S is a DS but not capacitated. First, consider (i); if S is not a DS, there must be one or more vertex $i \notin S$ not having an IN neighbor. If all OUT neighbors of vertex i with a lower ID have a dominator, *IsFull* of all its IN neighbors are true, and having the least ID in its neighborhood makes R7 enable. R5 or R7 is enabled if there is at least one neighboring vertex with a lower ID in the OUT state. This circumstance yields a contradiction against the assumption, which claims that there is no enabled vertex. Secondly, j is a vertex in S that has an overflow capacity by considering (ii). This makes R2 enable. After that, no rule makes the capacity of $Dominatees_j$ overflow. Since S is a DS, a vertex out of S must have at least one IN neighbor. If the capacity of the neighbor vertex (j) not having a dominator of vertex i is not full, it makes R3 enable. When the capacity of vertex j is full, i or one of its OUT neighbors not having a dominator neighbor vertex with a lower ID executes R7. Now, a contradiction arises against our assumption. Lemma 1 supports a pair of dominators and dominatee to match correctly; there is no rule to be enabled. Consequently, our theorem holds. ■

B. CONVERGENCE

Lemma 2: *IsFull* flag of a vertex in the IN state is precisely correct as a result of the first move.

Proof: Throughout the first move, all the vertices that are in IN states get to know the dominators of all of their OUT neighbors. Let j be a vertex in IN state; if the cardinality of $Dominatees_j$ is identical to the capacity value, vertex j is the dominator of all OUT neighbors in set $Dominatees_j$, so the statement "$IsFull_j = \text{false}$" changes to true. Otherwise, it remains incorrect. After the first move, there is no rule to let the $IsFull_j$ be false. ■

Lemma 3: R8 can be enabled at most twice, where the first occurrence must constitute the first move, and the last occurrence must constitute the last move.

Proof: Let j be a vertex in IN state in G. If $IsFull_j$ is correct and set to true within the initial configuration, then no rule makes j execute any of the rules. In order for a dominatee vertex i in $Dominatees_j$ can run R6, j must run R3 to delete i from its $Dominatees_j$. Nevertheless, i must run R6 so that j may run R3. Hence, there is an obvious deadlock concerning i and j. In this situation, if $IsFull_j$ is correct and set to true in the initial move, it remains unchanged. Now assume that $IsFull_j$ is incorrect at the beginning and node j does not run any rule among all rules, but R8, then the vertex j can execute R8, and $IsFull_j$ will be correct after the first move as shown by Lemma 2. If $IsFull_j$ is set to true after the initial move, it remains so. If node j runs R8 in any step following the first move, it cannot run any rule. Because when the capacities of vertices in $Dominatees_j$ are full, and all dominatees in $Dominatees_j$ select j as the dominator, it sets $IsFull_j$ to true (from false). There is not any rule that lets $IsFull_j$ be false again. Therefore, we prove that a vertex can execute R8 at most twice so that the first one should be the initial move, and the last one should be the final move. ■

Lemma 4: A vertex may run R2 at most once and as its initial move.

Proof: Since the first configuration of the system may be in any state, it is possible that the cardinality of $Dominatees_j$ of IN vertex j can initially have an overflow capacity in the initial state. Consequently, R2 can be run for the initial move, and it does not make the number of elements in $Dominatees_j$ overflow the current capacity. Following that, no rule is designed to force the capacity of $Dominatees_j$ to overflow. Therefore, a vertex has a mere opportunity to run R2 once within its first move. ■

Lemma 5: Any vertex may enable R1 at most once and R7 at most twice. During the execution of A_{CapDS} under an unfair distributed scheduler, solely the following two state sequences (as well as their suffixes) are possible:

"OUT - IN - OUT"

"OUT - IN - OUT - IN"

Proof: Assume that $Dominator_i$ of a vertex i in the OUT state is initially *NULL*, *IsFull* variables of all IN neighbors having lower IDs are true and incorrect, and all OUT neighbors having lower IDs have a dominator. In this situation, i runs R7 to change its state to IN (creating an "OUT - IN" sequence), so any IN neighbor having a lower ID runs R8 and switches its *IsFull* to false. Vertex i runs R1 as the second move and makes a state transition to OUT again. Hence, vertex i creates the sequence "OUT - IN - OUT". For executing R7 one more time, all IN neighbors of vertex i with lower IDs must run R8 and set their *IsFull* to true, and all OUT neighbors having lower IDs should select their dominator by running R5. In this situation, i runs R7 for the second time and does not make a transition. Since IN neighbors do not run any rule after the second R8, as of Lemma 3. Conversely, even if any OUT neighbor with a lower ID sets the dominator to *NULL* by R6, it cannot impose vertex i to execute R1. Vertex i has a "OUT - IN - OUT - IN" sequence after these phases.

Now suppose that the vertex i initially is in IN state; if there exists any neighboring vertex j with a lower ID in the IN state, and has $IsFull_j = false$, then vertex i executes R1 to make a transition to the OUT state. If vertex i does not run R7 and any IN neighbor j accumulates vertex i to $Dominatees_j$, vertex i then executes R5 without executing another rule. After these steps, i creates a state sequence of "IN - OUT". ■

Lemma 6: R5 and R6 may be run at most n^2 times depending on whether the system state is steady.

Proof: In the initial state, assume that vertex i is in the OUT state, so it can run R7, R6, or R5. If the $Dominator_i$ does not equal to $NULL$ and the vertex i is not residing in the dominatees of $Dominator_i$, R6 will be enabled in the first move and lets $Dominator_i$ be $NULL$. Later on, if an IN neighbor j of vertex i included i in its $Dominatees_j$, R5 is executed by vertex i. Then, vertex j changes its state to OUT by executing R1. Under such a condition, vertex i runs R6 one more time. R6 and R5 rules can be run as a loop during i holds IN neighbors, keeping i as a member of their *Dominatees* set and executing ("IN - OUT") move with R1. On the other hand, there must be one IN neighbor k in the system, and k remains in the IN state as long as its ID is the smallest among its IN-state neighbors. When vertex k assigns vertex i a member of $Dominatees_k$ and vertex i sets k as its dominator, it is unable to run any rule since no rule may break the match-up of vertices i and k.

As $Dominatees_k$ does not include i, an OUT neighbor m had been running R1 before can later execute R7 and makes i a member of $Dominatees_u$ by R3. Moreover, m cannot execute R1 again as directed by Lemma 5. So, u becomes the dominator of i. Vertex i cannot execute any of the rules, breaking the match-up between i and m. Vertex i can make an "OUT - IN - OUT" sequence and remains as is per Lemma 6. From here, the cumulative move count for R6 and R5 must be multiplied by two. The equation given below presents the maximum possible number of moves of R6 and R5, where n is the member vertices of a graph G:

$$
\begin{aligned}
&= 4xy(x = |\{S_i = OUT\}|, y = |\{S_j = IN\}|, n = x + y)\\
&= 4x(n - x)\\
&= -4x^2 + 4nx
\end{aligned}
$$

$x_{max} = \frac{n}{2}$ *and* $4\frac{n^2}{4} = n^2$ is clearly the largest move count for the system-of-interest. ■

Lemma 7: Until the system state becomes steady, rules R3 and R4 may be enabled at most $\frac{2n^2}{3}$ times.

Proof: An IIoT system can be started from an arbitrary unstable state or configuration, as mentioned earlier. Assume that there is initially an unstable system that consists of n vertices. There is a vertex i that is in IN state. Let D be the dominator vertices set in CapDS, E be the dominatee vertices set; further, $|D| = d$ and $|E| = e$. Each vertex in E has at least one neighbor vertex in CapDS. Now consider the first move: d vertices can run R3, and all of them may put the same vertex into their *Dominatees* set. At least one vertex in E turns R5 enable as the second move. Therefore, each vertex in D has a capacity of 1, the same in the worst case. Then, (d-1) vertices can run R4 and delete the matched vertex among E from their *Dominatees* set as the third step. For the fourth move, (d-1) vertices can run R3 and write the same vertex into their *Dominatees* set from E except for the matched vertex. At least one vertex in E makes R5 enable and selects a dominator from D for a fifth move. By the way, the vertex i can make an "IN - OUT - IN" state sequence and remains so by Lemma 5. After that, the total number of moves for R3 and R4 should be multiplied by two. The equations below prove that the dominators can run at most R3 and R4 until the system enters a steady state.

$$
= \begin{cases} d + 2\sum_{i=1}^{d-1}(d-i), & d \le e \quad (1)\\ d + 2\sum_{i=1}^{e-1}(e-i) + d - e, & d > e \quad (2)\end{cases}
$$

Case 1: if $d \le e$

$$
\begin{aligned}
d + 2\sum\nolimits_{i=1}^{d-1}(d-i) &= d + 2(\sum\nolimits_{i=1}^{d-1} d - \sum\nolimits_{i=1}^{d-1} i)\\
&= d^2
\end{aligned}
$$

$d_{max} = \frac{n}{2}$ and $\frac{2n^2}{4} = \frac{n^2}{2}$ is the greatest move count.

Case 2: if $d > e$

$$
\begin{aligned}
&d + 2\sum\nolimits_{i=1}^{e-1}(d-i) + d - e\\
&= d + 2(\sum\nolimits_{i=1}^{e-1} d - \sum\nolimits_{i=1}^{e-1} i) + d - e\\
&= 2(n-e)e - e^2\\
f(y) &= 2ne - 3e^2\\
f(e) &= 2n - 6e = 0
\end{aligned}
$$

$y_{max} = \frac{n}{3}$ and $\frac{2n^2}{3}$ is the highest move count. Due to $\frac{n^2}{2} < \frac{2n^2}{3}$ for $n \ge 0$, thus R3 and R4 can be executed maximum $\frac{2n^2}{3}$ times until the system enters in steady state. ■

Theorem 2: A_{CapDS} is considered self-stabilizing under the command of an unfair distributed scheduler. Furthermore, it converges to a stable state, including a CapDS after at most $(\frac{5n^2}{3}+6n)$ moves.

Proof: Per Lemma 5, a vertex may run R1 at most once and R7 twice. That limitation yields at most $3n$ moves. As in Lemma 4, a vertex may run R2 at most once. Thence, it yields at most n moves. Lemma 7 allows at most $\frac{2n^2}{3}$ moves owing to R3 and R4. R5 and R6 can be run at most n^2 times as explained in Lemma 6, whereas Lemma 3 proves that R8 may be run twice by any vertex; therefore, at most $2n$ times by n vertices. Ultimately, the total move count can be found as $3n + n + \frac{2n^2}{3} + n^2 + 2n = \frac{5n^2}{3} + 6n$. ■

Theorem 3: A_{CapDS} yields a solution for the CapDS problem with an approximation ratio of 6, given that a UDG model is accepted and all vertices have a uniform capacity.

Proof: Let us assume that the set D^* stands for the minimum CapDS of V of a UDG G. So the size of D^* can be formulated as $|D^*| = V/(c+1)$. Also, let S be the CapDS generated by A_{CapDS}. Assume that S' is the set of vertices in S whose degrees are at least equal to the capacity, and S'' is the set of other vertices in S. The vertices in S' can dominate a maximum of c vertices. The vertices in S'' create a MIS. Because, it is not possible for IN vertices in S'' to be neighbors of each other, and every OUT vertex already has

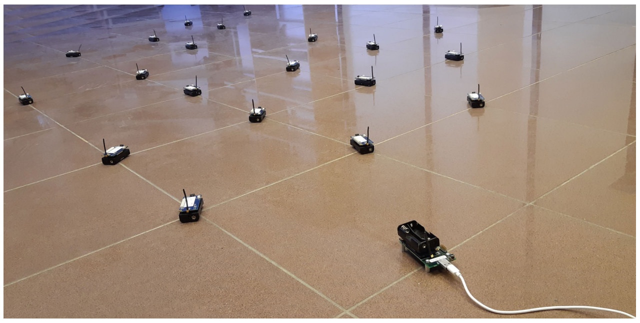

Fig. 2. An example IIoT setup with IRIS motes as the testbed.

an IN neighbor existing in S''. *IsFull* variable of an IN vertex in S'' cannot be true because of its degree being lower than the capacity. Any vertex in MIS can dominate at most five vertices in G. As $S^{'} \leq D^*$ and $S'' \leq 5D^*$, and $S \leq 6D^*$, A_{CapDS} has 6 approximation ratio. ■

V. PERFORMANCE EVALUATION

We have evaluated the performance of the proposed algorithm both via testbed experiments and simulations.

A. EXPERIMENTS ON TESTBED

A thorough experimental evaluation of A_{CapDS} was done on a testbed system that includes IRIS wireless sensor nodes in our laboratory environment. We generated numerous topologies with node cardinalities ranging from 10 to 40 by augmenting 10 nodes in each trial. The created topologies were undirected, and each node was assigned a unique ID. The topologies used in the experiments were sparse, medium, and dense, with degrees of four, six, and eight, respectively. The number of moves, received byte counts, and energy consumption were calculated as cost indicators Our test scenario involves randomly killing 20% of the existing nodes in each experiment round. Those experiments were iterated 20 times for each measurement, and the mean averages were recorded. All the nodes initially sent a *Hello* packet to their neighbors immediately at a 1-hop distance in order to announce their initial states. If any node i is a dominator, it sends its id_i, S_i, $IsFull_i$ and $Dominatees_i$ parameters in a message, whereas a dominatee j sends its id_j, S_j and $Dominator_j$ parameters. Each node attached to the network was able to run the algorithm at the same moment. If a rule's preconditions were all satisfied, the node could move by executing its rule(s). When a node changed its state, it broadcasted its new state, allowing neighbors to be acknowledged. When no enabled node was left, the network was stabilized, and a CapDS was created. It translates as each dominatee found a dominator, and each dominator constituted a set of dominatees whose size did not exceed the available capacity.

The energy utilization for each node is deduced by accounting for the bytes transmitted (S) and received (R). The IRIS motes, operating at a data transmission rate of 250 kbps-which corresponds to 31.25 kB/s-draw a current of approximately 16 mA in the receive mode and 17 mA in the transmit mode at a transmission power of TX = 3 dBm. With an operational voltage of 3300 mV, the energy expenditure (E) is estimated using the fundamental electrical power equation $E = V \times I \times T$. Hence, the energy consumed (E) is approximated by the formula:

$$E \approx \left(\frac{S \times 17 + R \times 16}{31.25} \right) \times 3.3 \text{ mJ}$$

Figure 2 provides a visual representation of our testbed platform, which is designed to simulate a typical IIoT deployment. This platform comprises 40 IRIS motes, which are high-performance, IEEE 802.15.4-compliant wireless sensors. These motes are strategically distributed to mimic the real-world distribution of devices in an IIoT network. Each mote is equipped with a variety of sensors to collect data, and they communicate with each other to form a mesh network, reflecting the interconnected nature of IIoT systems. The testbed also includes a sink device, which serves as the central point of data collection and communication. This device is connected to a PC, allowing for monitoring, controlling, and analyzing the data collected from the motes. The sink device plays a crucial role in managing the network, as it is responsible for tasks such as data aggregation, network routing, and the coordination of the motes.

Fig. 4 shows the (average) number of moves in a run of A_{CapDS} per varying node counts and densities. Per definition, the number of moves directly affects the received data size (i.e., number of bytes) and eventually the energy consumption as each node declares its new state info to its neighbors more often (as states change). Clear from the figure is that when the node cardinality increases, the number of required moves increases linearly. Besides, the move counts are generally stable against varying node densities. Fig. 4 illustrates the received number of bytes during A_{CapDS} against the node cardinality and also density. The average node degree is larger in denser topologies as the interconnections between the nodes are more in number. Per the measurements, the received byte count positively correlates with the number of nodes and the node densities. Fig. 5 provides an insight into the energy consumption of running A_{CapDS} with varying node numbers and densities. The energy consumption measurements yielded results similar to those of the received byte counts. That may not be a surprise as it has a crucial contribution to the system's total energy consumption. Thence, the energy consumption scales up or down proportionally to the number of nodes involved and also their topological density. These experimental results of IRIS nodes suggested that our algorithm consumes a reasonable amount of resources, and this behavior is quite stable even when node counts and degrees vary.

As no distributed self-stabilizing capacitated DS approaches can be found (as discussed in the related works), two CapDS algorithms were derived from the self-stabilizing DS algorithm that utilizes $4n$ moves proposed by Chiu et al. [42] by implementing the hierarchical collateral

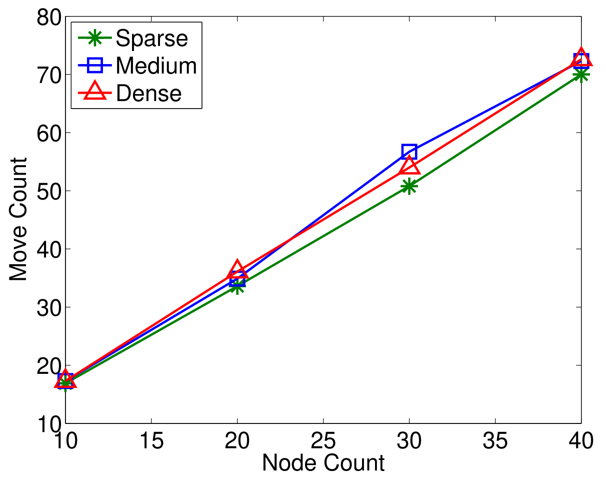

Fig. 3. **Move count of A_{CapDS} vs node count and density.**

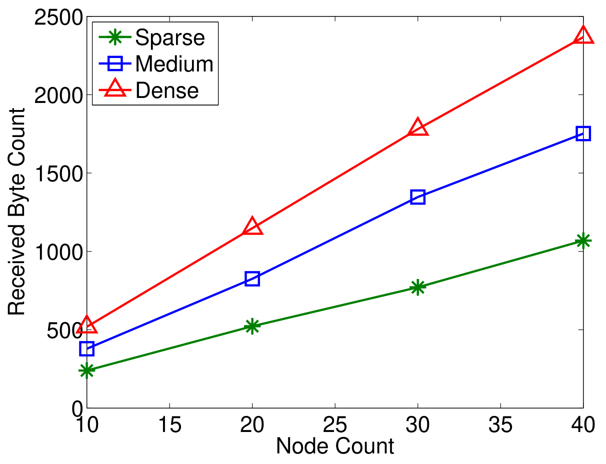

Fig. 4. **Number of received bytes in A_{CapDS} vs node count and density.**

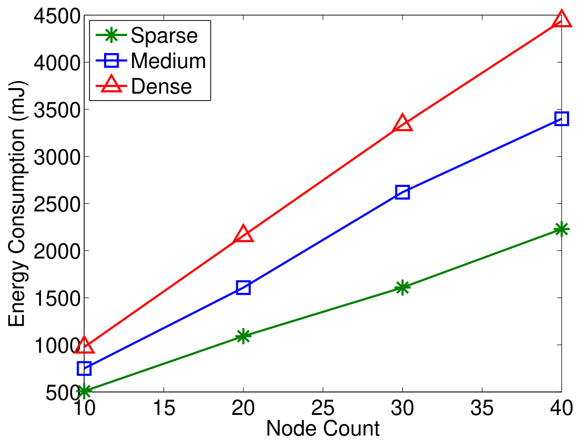

Fig. 5. **Energy consumption of A_{CapDS} vs node count and density.**

TABLE 3. **Move count comparison of A_{CapDS}, C_{Random}, and $C_{ID-based}$ algorithms for real experiments.**

Node Count	Move Count		
	A_{CapDS}	C_{Random}	$C_{ID-based}$
10	22.30	44.60	44.50
20	53.90	107.80	107.70
30	74.60	149.20	149.10
40	102.70	205.40	205.20

TABLE 4. **Energy consumption comparison of A_{CapDS}, C_{Random}, and $C_{ID-based}$ algorithms for real experiments.**

Node Count	Energy Consumption (mJ)		
	A_{CapDS}	C_{Random}	$C_{ID-based}$
10	1164.10	2328.20	2327.90
20	2733.87	5467.74	5467.34
30	3754.09	7508.18	7507.80
40	5162.11	10324.22	10323.30

composition method (as used in [55]). C_{Random} denotes the first algorithm, which includes a randomized approach where the dominatees determine their dominator randomly, while the dominator nodes implicitly write the dominatees into their set of dominatees if the node runs no rule. $C_{ID-based}$ stands for the second algorithm, which is based on a minimum ID priority-based approach where the dominatees always determine the dominator with the minimum ID, and the dominators put them implicitly in their dominatees sets if the node runs no rule.

The relationship between move count and network longevity is inversely proportional, as a higher number of moves often translates to increased energy depletion. The data presented in Table 3 underlines this relationship, showcasing a linear increment in move count in tandem with node count increments. This emphasizes the necessity for algorithms that can minimize move count to extend the operational lifespan of the network. Our analysis indicates that A_{CapDS} outperforms both C_{Random} and $C_{ID-based}$ in terms of efficiency.

Energy consumption is a vital metric for assessing the efficiency of distributed systems, particularly in the context of wireless sensor networks. In Table 4, we examine the energy expenditure in correlation with varying node counts. As node count escalates, so does the energy consumption, which is expected due to the elevated operational demands. Notably, A_{CapDS} demonstrates a substantial improvement in energy conservation, with results that are significantly lower than those of C_{Random} and $C_{ID-based}$. These findings are consistent across different node densities, highlighting A_{CapDS}'s adaptability to diverse network structures. Consequently, A_{CapDS} exhibits a robust potential for prolonging network lifetime by ensuring minimal energy consumption compared to its counterparts, which is a critical consideration in the design and application of algorithms in energy-constrained environments.

B. SIMULATIONS

Throughout the testbed experiments, we were limited to 40 sensor nodes that were available in the laboratory setup. In order to extend our measurement efforts to a

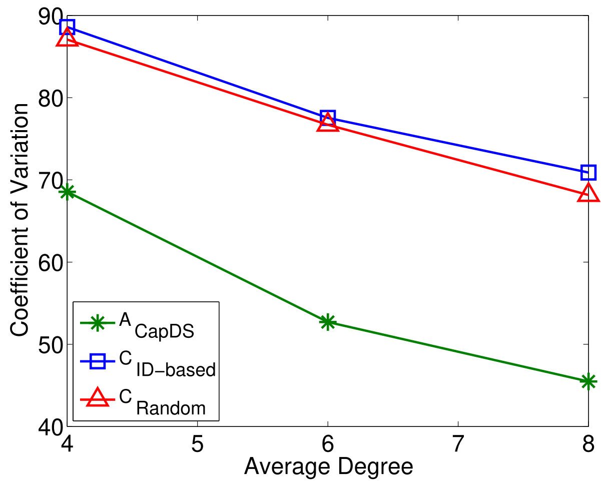

Fig. 6. **Coefficient of variations of the algorithms vs Node count.**

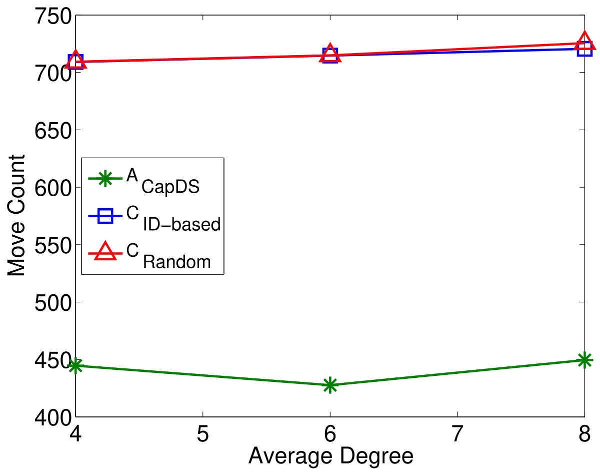

Fig. 9. **Move count of the algorithms vs average degree.**

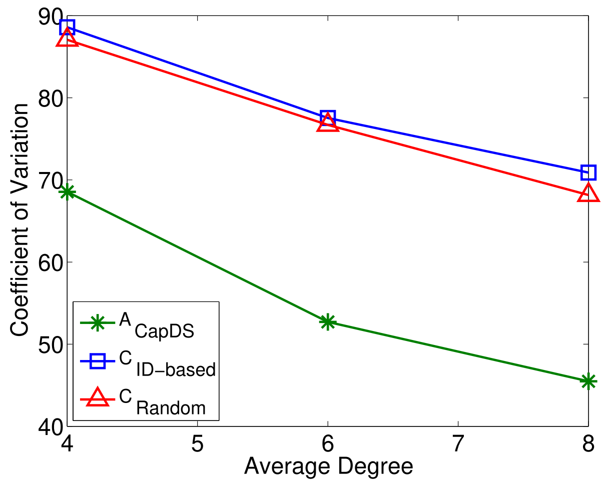

Fig. 7. **Coefficient of variations of the algorithms vs Average degree.**

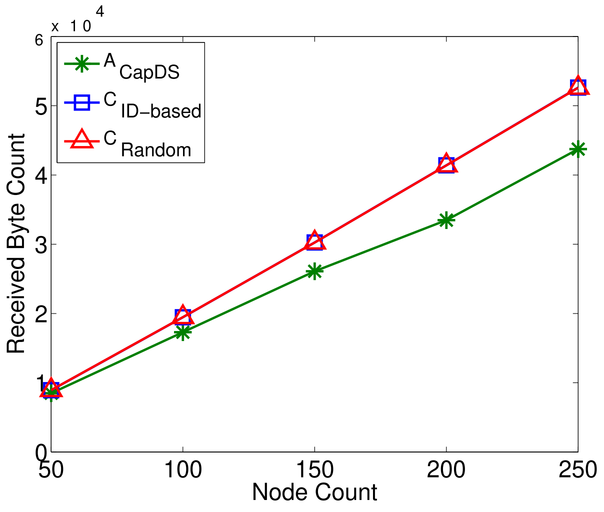

Fig. 10. **Received byte count of the algorithms vs node count.**

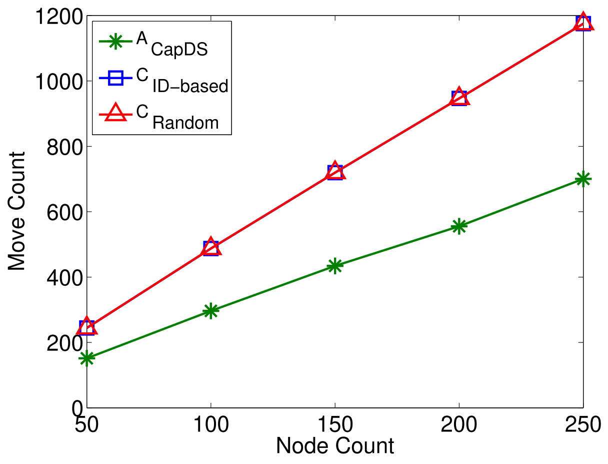

Fig. 8. **Move count of the algorithms vs node count.**

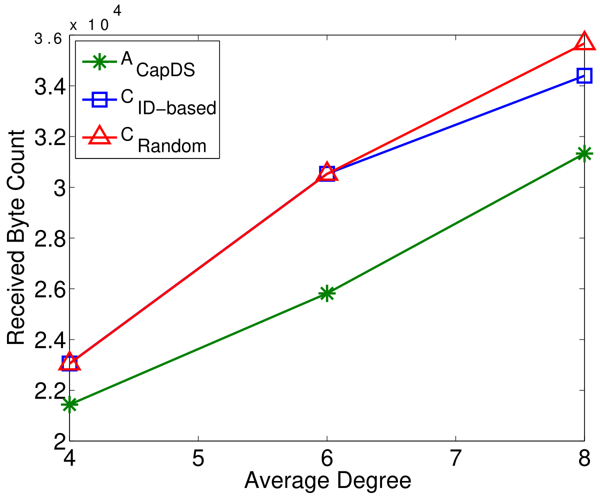

Fig. 11. **Received byte count of the algorithms vs average degree.**

much larger scale, we also made simulations, which not only allowed us to use more crowded network topologies.

In addition to the real-world experiments, the described methodology for calculating energy utilization was also employed in simulation scenarios. The simulation parameters were aligned with those of the IRIS motes used in the empirical studies. These parameters included a transmission rate of 250 kbps, corresponding to a real-time data throughput of 31.25 kB/s. Similarly, the simulated nodes were configured to reflect the power consumption characteristics of the IRIS motes, with a current draw of 16 mA in receive mode and 17 mA in transmit mode, operating at a transmission power level of 3 dBm. The operational voltage was set to 3300 mV, consistent with the hardware specifications. By adhering to these parameters, the simulations were able to accurately emulate the energy expenditure calculated using the formula $E = V \times I \times T$, thus ensuring the validity and comparability of the simulation results with the actual energy consumption observed in the real-world deployments. This approach enabled a robust analysis of the energy dynamics, offering relevant and applicable insights to both theoretical and practical applications in the field.

All three topologies, being A_{CapDS}, C_{Random} and $C_{ID-based}$ were implemented and tested on TOSSIM discrete event simulator. Undirected and random graphs were generated to form the topologies. The topologies were set to change from 50 to 250 nodes in each trial (with a step size of 50). Like in the testbed experiments, the topological densities were divided into three classes: sparse, medium, and dense, where the average node degrees were (approximately) four, six, and eight, respectively. Each measurement was recorded as a mean average of 30 repeated runs. Also presented are a comparison of those algorithms per varying node cardinalities and the average degrees in terms of coefficient of variation (CV), number of moves, number of received bytes, and energy consumption, again as done during the real experiments. The CV is a measure of relative variability and is expressed as $CV = (Standard\ Deviation/Mean) \times 100$. It is used to render the balance of those clusters shaped by the cluster heads (dominators) and the members (dominatees).

Fig. 6 demonstrates the comparison between the CV of the algorithms per varying node cardinalities. As seen in the figure, if the node cardinality increases, then CV decreases. That is because the dominatees have more dominators among neighbors to interfere within the assignment that pushes to build more balanced clusters, provided that the number of nodes scales up. Among all,the CV of A_{CapDS} is the smallest resulting that A_{CapDS} is on average 49% better than C_{Random} and 55% better than $C_{ID-based}$. Similarly, Fig. 7 shows the CV values of the algorithms against varying average node degrees. If the degree is increased, the size (i.e., cardinality) of DS decreases, leading to a reduction in the CV values. A_{CapDS} performs decisively better than its competitors that have similar results, as shown in Fig. 6. To sum up, it is safe to state that the CV values of A_{CapDS} are significantly better than its competitors per varying number of nodes and node degrees.

The number of moves has critical importance in the course of energy consumption. The number must be minimized in order to enable a longer network lifetime. As illustrated in Fig. 8, a positive linear correlation exists between the move counts and the number of nodes in the network. Furthermore, A_{CapDS} provide 61% better results than C_{Random} and 60% better than that of $C_{ID-based}$. In Fig. 9, the number of moves of algorithms are depicted per the average node degrees, where A_{CapDS} promises the best stats among others regardless of the subject topology. All in all, A_{CapDS} clearly requires fewer moves when compared to other options.

The received number of bytes per node count was shown in Fig. 10 for all algorithms. IT can be observed that the received byte counts of the algorithms increase mostly linearly when the number of nodes increases. A_{CapDS} has a clear advantage as the data traffic it causes is 13% lower than that of C_{Random} and 9% lower than $C_{ID-based}$. The relation between the node degrees and the number of received bytes is given in Fig. 11. Again, there is an observable linear correlation, where A_{CapDS} is the best performer.

The key to prolonging the network lifetime in IIoT is obviously energy efficiency. Fig. 12 shows the energy efficiency of the algorithms by depicting the correlation between the estimated consumed energy and the number of nodes in the topology. With all algorithms, the energy consumption tends to increase if the number of nodes increases. From the figure, A_{CapDS} has the best energy efficiency, as it consumes 14% less energy than C_{Random} and 10% less energy than $C_{ID-based}$. Fig. 13, on the other hand, presents the energy consumption per varying average node degrees. A_{CapDS}, again, has the lowest energy consumption for any given average node degree. Eventually, the measurements from the simulations suggest that A_{CapDS} outperforms its competitors in terms of CV, number of required moves, number of received bytes, and the energy consumption per varying number of involved nodes and node densities.

A dominator's capacity is said to be uniform (resp. non-uniform) if all nodes in the graph share the same (resp. variable) capacity value. A_{CapDS} is able to work with both uniform and non-uniform capacities, but it can provide the 6-approximation ratio only for the case of uniform capacities. For homogeneous IIoT deployments, uniform capacity is highly appropriate. Nevertheless, heterogeneous networks can generally be seen as more realistic. Wireless communication is the major energy-consuming action in IIoT [56]. The network lifetime in an IIoT deployment is often described as the time span between when the experiment begins until the first node failure in a dominating set (due to energy outage, etc.) [56]. We calculated the network lifetime of algorithms against the number of included nodes and the average node degrees. Note that the size of data that can be sent in a mere packet is limited to 127 bytes per to packet structure defined by IEEE 802.15.4 standard. E_i and E_{max} represent the energy (in mJ) of the node i and the maximum consumed energy related to a 127 bytes-sized packet, respectively.

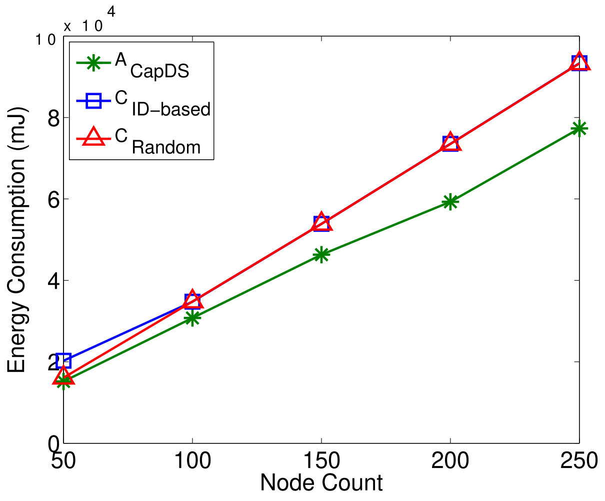

Fig. 12. **Energy consumption vs node count.**

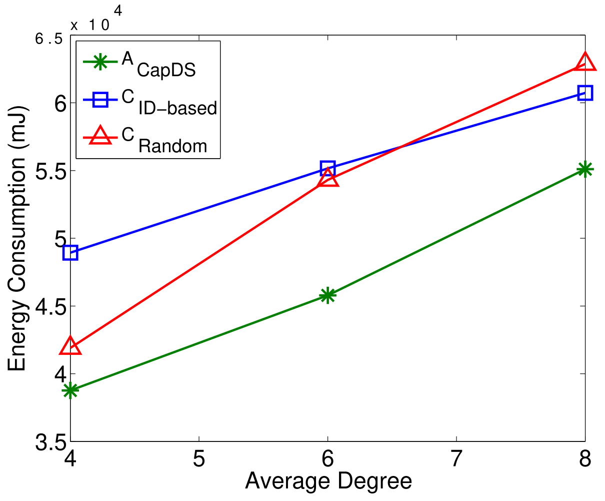

Fig. 13. **Energy consumption vs average node degree.**

The nodes were initially assigned a random amount of energy in the range between 1000 × E_{max} and 10000 × E_{max} mJ. It is assumed that when an algorithm constructs a CapDS, each dominatee of a dominator sends a 127 bytes-long packet in each round. The capacity of the node i is denoted as c_i, and the degree of that node is denoted D_i. Moreover, D_{avg} stands for the degree of the neighbors of node i, and E_{avg} is the average energy level in mJ. The capacity of the nodes in the IIoT deployment is initially calculated with the formula below:

$$c_i = \begin{cases} \left\lceil \dfrac{E_i \times D_{avg}}{E_{avg}} \right\rceil, & \text{if } \left\lceil \frac{E_i \times D_{avg}}{E_{avg}} \right\rceil \leq D_i \qquad (3) \\ D_i, & \text{if } \left\lceil \frac{E_i \times D_{avg}}{E_{avg}} \right\rceil > D_i \qquad (4) \end{cases}$$

Fig. 14 illustrates the network lifetime offered by all three algorithms per varying number of nodes in the network. Generally speaking, if the node cardinality increases, the time to the first node failure reduces. Because, if the number of nodes increases, the likelihood that a node with little energy

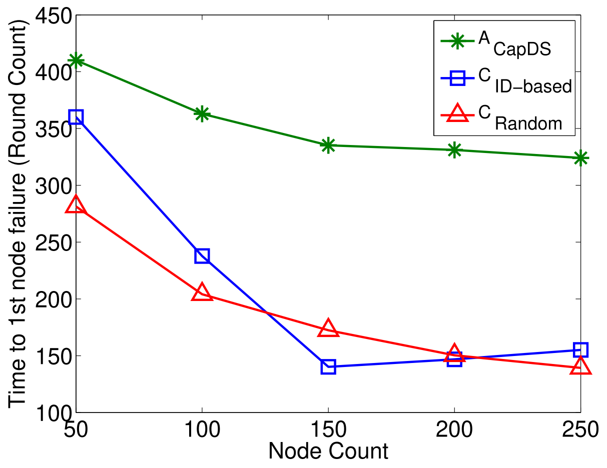

Fig. 14. **Network lifetime of the algorithms vs node count.**

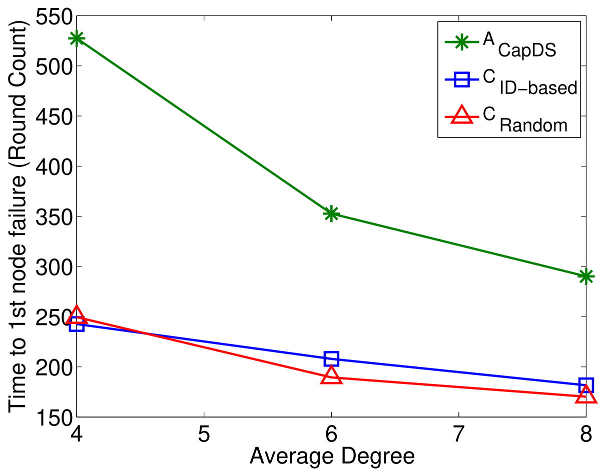

Fig. 15. **Network lifetime of the algorithms vs a dominator's capacity is said to be uniform degree.**

resources being a dominator increases as well. Even so, A_{CapDS} has the longest network lifetime as it endures A_{CapDS} 86% longer than C_{Random} and 69% longer than $C_{ID-based}$. The network lifetimes per average node degree are shown In Fig. 15. A network's node density may affect the degrees of individual nodes, and eventually the size of CapDS. Therefore, the time to the first node failure drops remarkably if the average degree is increased. In par with the previous results is that A_{CapDS} is the best performing one among all in terms of network lifetime. An apparent minor drawback for A_{CapDS} can be its exaggerated susceptibility to increase in the node degrees; though it is still better than alternatives, working in denser networks may require special handling. Overall, these results assert that A_{CapDS} runs successfully and prolongs the network lifetime to a large extent, even if the capacity is non-uniform.

VI. CONCLUSION

DS is a fundamental structure for clustering and routing in IIoT, where the research threads related to constructing DS and its variants are still hot in many aspects. CapDS is a constrained version of DS in which each dominator is matched with a bounded number of dominatee. This strategy is significant for preserving the residual energy of dominators due to prolonging the network lifetime. Energy is not the only concern for these networks since IIoT is composed of tiny sensor motes that are prone to failures and environmental effects. Thus design, analysis, and implementation of self-stabilizing protocols aiming at fault-tolerance as well as energy efficiency are of utmost importance.

This paper proposes the first distributed self-stabilizing CapDS algorithm for energy-efficient and fault-tolerant IIoT. To prove the correctness of the proposed algorithm, we provide a converge and closure analysis of the proposed algorithms. We show that the move count of the proposed algorithm is bounded by $\frac{5n^2}{3}+6n$. We theoretically prove that the proposed algorithm has a 6-approximation ratio for UDG-modeled IIoT.

We implement the algorithm in a testbed of 40 IRIS sensor motes to measure its performance in real sensor motes. To evaluate the proposed algorithm in large-scale settings, we implemented it in the TOSSIM simulator and tested its performance against various node counts and average node degrees. Extensive experiments realized on a testbed, as well as the simulations, reveal that the algorithm we proposed in this paper is favorable in terms of reducing the number of required moves, balancing clusters, consuming energy, and prolonging network lifetime.

Future work could address several open problems in the context of the IIoT, including designing an approximation algorithm that can operate effectively in IIoT environments with non-uniform capacities. Additionally, research could focus on estimating the expected number of moves and reducing the total move counts required in worst-case scenarios. These advancements could further enhance the efficiency and adaptability of the CDS algorithm in complex IIoT settings.

REFERENCES

[1] N. N. Srinidhi, S. M. D. Kumar, and K. R. Venugopal, "Network optimizations in the Internet of Things: A review," *Eng. Sci. Technol., Int. J.*, vol. 22, no. 1, pp. 1–21, Feb. 2019.

[2] Z. A. Dagdeviren, "Weighted connected vertex cover based energy-efficient link monitoring for wireless sensor networks towards secure Internet of Things," *IEEE Access*, vol. 9, pp. 10107–10119, 2021.

[3] R. Vijay, V. Ramasamy, M. Selvaraj, R. Anbazhagan, and A. Rengarajan, "Development of low-profile spectral signature chipless flexible RFID prototype for 5G supply chain IoT applications," *Eng. Sci. Technol., Int. J.*, vol. 36, Dec. 2022, Art. no. 101296.

[4] H. C. Altunay and Z. Albayrak, "A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks," *Eng. Sci. Technol., Int. J.*, vol. 38, Feb. 2023, Art. no. 101322.

[5] Z. A. Dagdeviren, "A metaheuristic algorithm for vertex cover based link monitoring and backbone formation in wireless ad hoc networks," *Exp. Syst. Appl.*, vol. 213, Mar. 2023, Art. no. 118919.

[6] X. Gong, D. Plets, E. Tanghe, T. De Pessemier, L. Martens, and W. Joseph, "An efficient genetic algorithm for large-scale transmit power control of dense and robust wireless networks in harsh industrial environments," *Appl. Soft Comput.*, vol. 65, pp. 243–259, Apr. 2018.

[7] E. El-shafeiy, K. M. Sallam, R. K. Chakrabortty, and A. A. Abohany, "A clustering based swarm intelligence optimization technique for the Internet of Medical Things," *Exp. Syst. Appl.*, vol. 173, Jul. 2021, Art. no. 114648.

[8] V. K. Akram, Z. A. Dagdeviren, O. Dagdeviren, and M. Challenger, "PINC: Pickup non-critical node based k-connectivity restoration in wireless sensor networks," *Sensors*, vol. 21, no. 19, p. 6418, Sep. 2021.

[9] Z. A. Dagdeviren, V. K. Akram, O. Dagdeviren, B. Tavli, and H. Yanikomeroglu, "K-connectivity in wireless sensor networks: Overview and future research directions," *IEEE Netw.*, vol. 37, no. 3, pp. 140–145, May/Jun. 2023.

[10] A. A. Al-Roubaiey, T. R. Sheltami, A. S. H. Mahmoud, and K. Salah, "Reliable middleware for wireless sensor-actuator networks," *IEEE Access*, vol. 7, pp. 14099–14111, 2019.

[11] O. Arapoglu, V. K. Akram, and O. Dagdeviren, "An energy-efficient, self-stabilizing and distributed algorithm for maximal independent set construction in wireless sensor networks," *Comput. Standards Interfaces*, vol. 62, pp. 32–42, Feb. 2019.

[12] S. Dolev, *Self-stabilization*. Cambridge, MA, USA: MIT Press, 2000.

[13] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Inf. Process. Lett.*, vol. 103, no. 3, pp. 88–93, Jul. 2007.

[14] M. T. Thai and D.-Z. Du, "Connected dominating sets in disk graphs with bidirectional links," *IEEE Commun. Lett.*, vol. 10, no. 3, pp. 138–140, Mar. 2006.

[15] C. U. Ileri and O. Dagdeviren, "A distributed and asynchronous approach for optimizing weighted graph matchings in wireless network services," *Exp. Syst. Appl.*, vol. 119, pp. 73–89, Apr. 2019.

[16] Q. Tang, K. Yang, J. Wang, Y. Luo, K. Li, and F. Yu, "Wireless sensor network MCDS construction algorithms with energy consideration for extreme environments healthcare," *IEEE Access*, vol. 7, pp. 33130–33144, 2019.

[17] T. Pino, S. Choudhury, and F. Al-Turjman, "Dominating set algorithms for wireless sensor networks survivability," *IEEE Access*, vol. 6, pp. 17527–17532, 2018.

[18] W.-H. Yang, Y.-C. Wang, and Y.-C. Tseng, "Efficient packet recovery using prioritized network coding in DVB-IPDC systems," *IEEE Commun. Lett.*, vol. 16, no. 3, pp. 382–385, Mar. 2012.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.

[20] Z. A. Dagdeviren, D. Aydin, and M. Cinsdikici, "Two population-based optimization algorithms for minimum weight connected dominating set problem," *Appl. Soft Comput.*, vol. 59, pp. 644–658, Oct. 2017.

[21] C. Luo, W. Chen, J. Yu, Y. Wang, and D. Li, "A novel centralized algorithm for constructing virtual backbones in wireless sensor networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2018, pp. 1–12, Dec. 2018.

[22] X. Sun, Y. Yang, and M. Ma, "Minimum connected dominating set algorithms for ad hoc sensor networks," *Sensors*, vol. 19, no. 8, p. 1919, Apr. 2019.

[23] L. Song, C. Liu, H. Huang, H. Du, and X. Jia, "Minimum connected dominating set under routing cost constraint in wireless sensor networks with different transmission ranges," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 546–559, Apr. 2019.

[24] D. Lisiecki, P. Zhang, and O. Theel, "CONE: A connected dominating set-based flooding protocol for wireless sensor networks," *Sensors*, vol. 19, no. 10, p. 2378, May 2019.

[25] T. Shi, S. Cheng, J. Li, H. Gao, and Z. Cai, "Dominating sets construction in RF-based battery-free sensor networks with full coverage guarantee," *ACM Trans. Sensor Netw.*, vol. 15, no. 4, pp. 1–29, Nov. 2019.

[26] J. Ali, J. Iqbal, S. Majeed, I. A. Mughal, A. Ahmad, and S. Ahmed, "Wireless sensor network design for smart grids and Internet of Things for ambient living using cross-layer techniques," *Int. J. Distrib. Sensor Netw.*, vol. 15, no. 7, Jul. 2019, Art. no. 155014771986220.

[27] Y. Guo, Y. Zhang, Z. Mi, Y. Yang, and M. S. Obaidat, "Distributed task allocation algorithm based on connected dominating set for WSANs," *Ad Hoc Netw.*, vol. 89, pp. 107–118, Jun. 2019.

[28] J. Liang, M. Yi, W. Zhang, Y. Li, X. Liang, and B. Qin, "On constructing strongly connected dominating and absorbing set in 3-dimensional wireless ad hoc networks," *Hindawi*, vol. 2020, pp. 1–12, Feb. 2020, doi: 10.1155/2020/9189645.

[29] A.-R. Hedar, S. N. Abdulaziz, A. A. Sewisy, and G. A. El-Sayed, "Adaptive scatter search to solve the minimum connected dominating set problem for efficient management of wireless networks," *Algorithms*, vol. 13, no. 2, p. 35, Feb. 2020.

[30] Z. A. Dagdeviren, "A pure genetic energy-efficient backbone formation algorithm for wireless sensor networks in industrial Internet of Things," in *Trends in Data Engineering Methods for Intelligent Systems*, J. Hemanth, T. Yigit, B. Patrut, and A. Angelopoulou, Eds. Cham, Switzerland: Springer, 2021, pp. 553–566.
[31] O. Arapoglu and O. Dagdeviren, "Distributed self-stabilizing capacitated maximal independent set construction in wireless sensor networks," *Wireless Pers. Commun.*, vol. 114, no. 4, pp. 3271–3293, Oct. 2020.
[32] O. Arapoglu and O. Dagdeviren, "A fault-tolerant and distributed capacitated connected dominating set algorithm for wireless sensor networks," *Comput. Standards Interfaces*, vol. 77, Aug. 2021, Art. no. 103490.
[33] G. W. Flake, S. Lawrence, and C. L. Giles, "Efficient identification of web communities," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2000, pp. 150–160.
[34] B. J. Kim, J. Liu, J. Um, and S.-I. Lee, "Instability of defensive alliances in the predator-prey model on complex networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 72, no. 4, Oct. 2005, Art. no. 041906.
[35] T. Haynes, D. Knisley, E. Seier, and Y. Zou, "A quantitative analysis of secondary RNA structure using domination based parameters on trees," *BMC Bioinf.*, vol. 7, no. 1, p. 108, Dec. 2006.
[36] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "What cannot be computed locally!" in *Proc. 23rd Annu. ACM Symp. Princ. Distrib. Comput.*, New York, NY, USA, Jul. 2004, pp. 300–309.
[37] F. Kuhn and R. Wattenhofer, "Constant-time distributed dominating set approximation," *Distrib. Comput.*, vol. 17, no. 4, pp. 303–310, May 2005.
[38] S. Kundu and S. Majumder, "A linear time algorithm for optimal k-hop dominating set of a tree," *Inf. Process. Lett.*, vol. 116, no. 2, pp. 197–202, Feb. 2016.
[39] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, "Self-stabilizing algorithms for minimal dominating sets and maximal independent sets," *Comput. Math. Appl.*, vol. 46, nos. 5–6, pp. 805–811, Sep. 2003.
[40] Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani, "A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph," in *Distributed Computing—IWDC*, S. R. Das and S. K. Das, Eds. Berlin, Germany: Springer, 2003, pp. 26–32.
[41] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, P. K. Srimani, and Z. Xu, "SELF-STABILIZING GRAPH PROTOCOLS," *Parallel Process. Lett.*, vol. 18, no. 1, pp. 189–199, Mar. 2008.
[42] W. Y. Chiu, C. Chen, and S.-Y. Tsai, "A 4n-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon," *Inf. Process. Lett.*, vol. 114, no. 10, pp. 515–518, Oct. 2014.
[43] M.-J. Kao and C.-S. Liao, "Capacitated domination problem," in *Algorithms and Computation*. Heidelberg, Germany: Springer, 2007, pp. 256–267.
[44] M. Dom, D. Lokshtanov, S. Saurabh, and Y. Villanger, "Capacitated domination and covering: A parameterized perspective," in*Parameterized and Exact Computation*. Heidelberg, Germany: Springer, 2008, pp. 78–90.
[45] H. L. Bodlaender, D. Lokshtanov, and E. Penninkx, "Planar capacitated dominating set is W[1]-hard," in *Parameterized and Exact Computation*, J. Chen and F. V. Fomin, Eds. Berlin, Germany: Springer, 2009, pp. 50–60.
[46] F. Kuhn and T. Moscibroda, "Distributed approximation of capacitated dominating sets," *Theory Comput. Syst.*, vol. 47, no. 4, pp. 811–836, Nov. 2010.
[47] M.-J. Kao and H.-L. Chen, "Approximation algorithms for the capacitated domination problem," in *Frontiers in Algorithmics*. Heidelberg, Germany: Springer, 2010, pp. 185–196.
[48] W. Shang and X. Wang, "Algorithms for minimum connected capacitated dominating set problem," *Discrete Math., Algorithms Appl.*, vol. 3, no. 1, pp. 9–15, Mar. 2011.
[49] M. Cygan, M. Pilipczuk, and J. O. Wojtaszczyk, "Capacitated domination faster than," *Inf. Process. Lett.*, vol. 111, nos. 23–24, pp. 1099–1103, Dec. 2011.
[50] A. Potluri and A. Singh, "Metaheuristic algorithms for computing capacitated dominating set with uniform and variable capacities," *Swarm Evol. Comput.*, vol. 13, pp. 22–33, Dec. 2013.
[51] M.-J. Kao, H.-L. Chen, and D. T. Lee, "Capacitated domination: Problem complexity and approximation algorithms," *Algorithmica*, vol. 72, no. 1, pp. 1–43, May 2015.
[52] M. Liedloff, I. Todinca, and Y. Villanger, "Solving capacitated dominating set by using covering by subsets and maximum matching," *Discrete Appl. Math.*, vol. 168, pp. 60–68, May 2014.
[53] A. Becker, "Capacitated dominating set on planar graphs," 2016, *arXiv:1604.04664*.
[54] R. Li, S. Hu, P. Zhao, Y. Zhou, and M. Yin, "A novel local search algorithm for the minimum capacitated dominating set," *J. Oper. Res. Soc.*, vol. 69, no. 6, pp. 849–863, Jun. 2018.
[55] A. K. Datta, L. L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre, "Self-stabilizing small k-Dominating sets," *Int. J. Netw. Comput.*, vol. 3, no. 1, pp. 116–136, 2013.
[56] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. New York, NY, USA: Wiley, 2007.

OZKAN ARAPOGLU received the M.Sc. and Ph.D. degrees in information technology from the International Computer Institute, Ege University. With a fervent focus on natural language processing and machine learning within the Internet of Things framework, he holds a prominent role in advancing artificial intelligence, providing training, and guidance on AI technologies in a leading tech firm known for its pioneering work in the field. His academic pursuits are broad, including distributed algorithms, the IoT, wireless sensor networks, graph theory, NLP, and ML.

UMUT CAN CABUK (Member, IEEE) received the B.Sc. degree in electronics engineering from Bursa Uludag University, Turkey, in 2012, the M.Sc. degree in information technology engineering from Aarhus University, Denmark, in 2015, and the Ph.D. degree from the International Computer Institute, Ege University, Turkey. He was a Research Assistant with Ege University. He is currently doing a postdoctoral studies with San Diego State University, USA. He has coauthored over 40 scholarly publications and has four patent applications. His research interests include drones, mobile and wireless networks, the Internet of Things, computer security, and graph theory.

ORHAN DAGDEVIREN (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from İzmir Institute of Technology and the Ph.D. degree from the International Computer Institute, Ege University, İzmir, Turkey. He is currently a Full Professor and the Head of the Network Engineering Science and Technology (NETOS) Laboratory, Department of Computer Engineering, Ege University. He has published more than 100 scholarly articles in respected conferences and journals. His research interests include distributed computing, fault tolerance, applied graph theory, and computer networking areas.

MOHARRAM CHALLENGER (Member, IEEE) received the Ph.D. degree in IT from the International Computer Institute, Ege University, in February 2016. From 2005 to 2009, he was a Faculty Member and a Lecturer with the Computer Engineering Department, IAU-Shabestar University. From 2010 to 2013, he was a Researcher and a Team Leader of the Bilateral Project between Slovenia and Turkey (TUBITAK). From 2012 to 2016, he was the Research and Development Director of UNIT IT Ltd., leading a national project funded by TUBITAK and two International Software-Intensive Projects in Europe called ITEA ModelWriter and ITEA Assume. From 2017 to 2018, he was a Faculty Member and an Assistant Professor with Ege University. From 2019 to 2020, he was a Postdoctoral Researcher with the University of Antwerp, working in Flanders Make Projects called PACo and DTDesign. He is currently a tenure-track Assistant Professor with the Department of Computer Science, University of Antwerp. His research interests include domain-specific modeling languages, multi-agent systems, cyber-physical systems, and the IoT.