T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

A STUDY ON THE ALGORITHMS FOR CAPACITATED DOMINATION PROBLEMS

PhD Thesis

Özkan ARAPOĞLU

International Computer Department

İzmir
2019

T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

A STUDY ON THE ALGORITHMS FOR CAPACITATED DOMINATION PROBLEMS

Özkan ARAPOĞLU

Supervisor: Assoc. Prof. Dr. Orhan DAĞDEVİREN

International Computer Department
Information Technology Third Cycle Programme

İzmir
2019

Özkan ARAPOĞLU tarafından Doktora tezi olarak sunulan “A Study on the Algorithms for Capacitated Domination Problems” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 06.08.2019 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu~~ ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	: Doç. Dr. Orhan DAĞDEVİREN	
Raportör Üye	: Prof. Dr. Mehmet Emin DALKILIÇ	
Üye	: Doç. Dr. Hasan BULUT	
Üye	: Dr. Öğr. Üyesi Fatih SOYGAZİ	
Üye	: Dr. Öğr. Üyesi Uras TOS	

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum "A Study on the Algorithms for Capacitated Domination Problems" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

06/08/2019

İmzası

Ö. Arapoğlu

Adı-Soyadı

Özkan ARAPOĞLU

ÖZET

KAPASİTE KISITLI HAKİMİYET PROBLEMLERİ İÇİN ALGORİTMALAR ÜZERİNE BİR ÇALIŞMA

ARAPOĞLU, Özkan

Doktora Tezi, Uluslararası Bigisayar Anabilim Dalı

Tez Danışmanı: Doç. Dr. Orhan DAĞDEVİREN

Ağustos 2019, 90 sayfa

Dağıtık sistemler, şeffaflık, açıklık, güvenilirlik, performans ve ölçeklenebilirlik içeren, ortak hedefleri başarabilmek için iş birliği içinde çalışan, otonom birbirine bağlı hesaplama elemanlarının toplamıdır. Dağıtık bir sistem, başlangıçta herhangi bir yasal olmayan durumdan başlamasına rağmen sınırlı zamanda yasal duruma kavuşursa ve dışsal bir müdahale olmadığı sürece öyle kalmaya devam ederse öz-kararlıdır. Kablosuz geçici ve sensör ağları (KGSA), herhangi bir altyapının yardımı olmaksızın binlerce kablosuz kendi kendine organize sensör düğümlerinden oluşan dağıtık ağlardır ve askeri gözetim, acil durum operasyonu, akıllı şehir, çevre bilimi ve hassas tarım gibi birçok gerçek dünya uygulaması için kullanılır.

Hakimiyet problemleri KGSA'lar gibi dağıtık sistemler için enerji etkinliği ve hata toleransı sağlamak için yaygın olarak kullanılır. Bunların uzantıları olan kapasite kısıtlı versiyonları ek olarak yük dengelemesi de sağlar. Bu tezde, bağımsız küme, hâkim küme ve bağlı hâkim küme kapasite kısıtlı hakimiyet problemleri için 3 dağıtık öz-kararlı algoritma önerdik. Bunların hepsi yakınsama ve kapalılık yönünden kanıtlandı. Ayrıca, test yatakları ile IRIS düğümler ve benzetimlerle TOSSIM üzerinde uygulandılar.

Anahtar sözcükler: Kapasite kısıtlı bağımsız küme, kapasite kısıtlı hâkim küme, kapasite kısıtlı bağlı hâkim küme, öz-kararlılık, dağıtık algoritmalar

ABSTRACT

A STUDY ON THE ALGORITHMS FOR CAPACITATED DOMINATION PROBLEMS

ARAPOĞLU, Özkan

PhD in International Computer Department

Supervisor: Assoc. Prof. Dr. Orhan DAĞDEVİREN

August 2019, 90 pages

Distributed systems are a collection of autonomous interconnected computing elements that cooperate to achieve common goals which include transparency, openness, reliability, performance, and scalability. A distributed system is self-stabilizing if it converges a legitimate state notwithstanding starting initially from any illegitimate state and stays so without any external intervention. Wireless ad hoc and sensor networks (WASNs) are distributed networks that consist of thousands of wireless self-organized sensor nodes without the aid of predefined infrastructure, and they are used for many real-world applications such as military surveillance, emergency operation, smart city, environmental science, and precision agriculture.

Domination problems are widely used to provide energy efficiency and fault tolerance for distributed systems such as WASNs. The capacitated versions which are extensions of them additionally provide load balancing. In this thesis, we propose three distributed self-stabilizing algorithms for capacitated domination problems which are independent set, dominating set, and connected dominating set. All of them are proven in terms of convergence and closure. Moreover, they are implemented on IRIS motes through testbeds and on TOSSIM through simulations.

Keywords: Capacitated independent set, capacitated dominating set, capacitated connected dominating set, self-stabilization, distributed algorithms

PREFACE

This thesis is submitted for the degree of Information Technology Doctor at Ege University, Turkey. It is original, unpublished, independent work by the author, O. Arapoglu. The research has been carried out at the International Computer Institute at Ege University. A period of six terms was spent from the proposal to the completion of the thesis. The research is supported by the Scientific and Technological Council of Turkey (TUBITAK) with ARDEB 1001 Project Grant (215E115).

The aim of this thesis is to designing capacitated distributed self-stabilizing algorithms for domination problems which are maximal independent set, dominating set, and connected dominating set. These domination problems are very important to provide energy efficiency in distributed systems such as wireless ad hoc and sensor networks. Although there are lots of works about domination problems, there are too few studies on capacitated versions of them. To the best of our knowledge, there are no capacitated distributed self-stabilizing algorithms for these three domination problems which are mentioned in detail in this work. This courage and excite me for writing my thesis. The thesis consists of seven chapters where the proposed algorithms are presented in chapters 4, 5, and 6. I hope that this thesis will be followed up by many researchers who study on capacitated domination problems.

IZMIR — Name-Surname

06.08.2019 — Özkan ARAPOĞLU

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (continuation)

Page

TABLE OF CONTENTS (continuation)

Page

TABLE OF CONTENTS (continuation)

TABLE OF CONTENTS (continuation)

Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES (continuation)

LIST OF FIGURES (continuation)

LIST OF SYMBOLS

Symbol	Explanation
V	The set of vertices
E	The set of edges
$G(V,E)$	A graph
n	The size of the vertices in graph G
i	Any node in graph G
id_i	The identifier of node i
c_i	The capacity of node i
S_i	The state of node i
N_i	The neighbors of node i within a one-hop distance
D_i	The degree of node i
E_i	The energy of node i
L_i	The lifetime of node i
λ	A configuration of a graph G consists of a tuple of all local states of the nodes
Γ	A set of all configurations
e	An execution of an algorithm
$\perp$	Null

LIST OF ABBREVIATIONS

Abbreviation	Explanation
WASN	Wireless ad hoc and sensor network
IS	Independent set
MIS	Maximal independent set
CapMIS	Capacitated maximal independent set
DS	Dominating set
CapDS	Capacitated dominating set
CDS	Connected dominating set
CapCDS	Capacitated connected dominating set
UDG	Unit disk graph
CH	Cluster head
CM	Cluster member
CV	Coefficient of variation
R	Rule
T_r	Transmission range

1. INTRODUCTION

A distributed system is a collection of autonomous computational nodes over a communication network that cooperates to accomplish common tasks. It supports the sharing of resources, distribution transparency, openness, and scalability for utilization. The importance of distributed systems is increasing day by day in our lives due to the recent technological improvements. The popular distributed system platforms are the Internet of the Things (IoT), grids, clouds, mobile ad hoc networks, and wireless ad hoc and sensor networks (WASNs). Design of algorithms for these systems called the distributed algorithms has become an excellent research area of computer science, engineering, applied mathematics, and other disciplines since they cope with difficult and complicated problems than the sequential algorithms.

WASNs are distributed networks that consist of thousands of wireless self-organized sensor nodes without a predefined infrastructure. WASNs are used for many real-world applications such as military surveillance, emergency operation, smart city, environmental science, and precision agriculture (Rashid and Rehmani, 2016; Jain et al., 2017; Ramson et al., 2017; Henry and Adamchuk, 2019). Scalability, fault tolerance, node deployment, and power management are some of the fundamental challenges in WASNs (Erciyes 2013). A sensor node consists of a sensor, microcontroller, memory, transceiver, and a battery. It has sense, communicating, and data processing functionalities. All the data collected by the sensor nodes are forwarded to a base station called a sink node (base station). The stored energy of the sensor nodes is meaningfully important due to a sensor node has generally non-rechargeable battery and restricted communication range in WASNs. The most energy is consumed by transceivers in order to send and receive messages between the sensor nodes. Therefore, a multi-hop communication approach is widely used due to support energy efficiency and load balancing in WASNs (Akyildiz et al., 2007; Singh and Sharma, 2015; Rostami et al., 2018).

The management and monitoring of the distributed systems are more complicated since they can be much larger and failed due to faults such as hardware malfunction, battery drain, and link failure. Fault tolerance is the ability to maintain desirable services without any interruption even though faults occur. It is classified as masking and non-masking. The masking fault tolerance supports the system service always available where the non-masking fault tolerance accepts a temporary inaccessible approach to the system service for a limited time. Self-stabilization is a non-masking fault tolerance approach. A distributed system is self-stabilizing if it reaches a legitimate state in a finite time and stays so without any external intervention despite starting from an arbitrary state (Dijkstra, 1974).

Domination problems which are independent set, dominating set, and connected dominating set are widely used to provide energy efficiency and fault tolerance for distributed systems such as WASNs. WASNs are widely modeled with unit disk graphs (UDGs) $G = (V, E)$ where V and E denote the set of nodes and the set of edges, respectively. Two nodes are neighbors if the Euclidean distance between these nodes is lower than the transmission range (T_r) of them (Clark et al, 1990). An example of WASN is presented in Figure 1.1. A subset S of V is an independent set if there are no neighbor nodes in S. The size of an independent set represents the number of nodes that it includes. A maximum independent set is the largest independent set. A maximal independent set (MIS) is an independent set which cannot be enlarged anymore. A dominating set (DS) of nodes D is a subset of V whose every node is either a member of D or a neighbor to a member of D. A connected dominating set (CDS) is a DS which induces a connected subgraph of G. Finding MIS, DS, and CDS of a graph is widely used for many important applications such as clustering, routing, data aggregation, topology control, and building other graph structures.

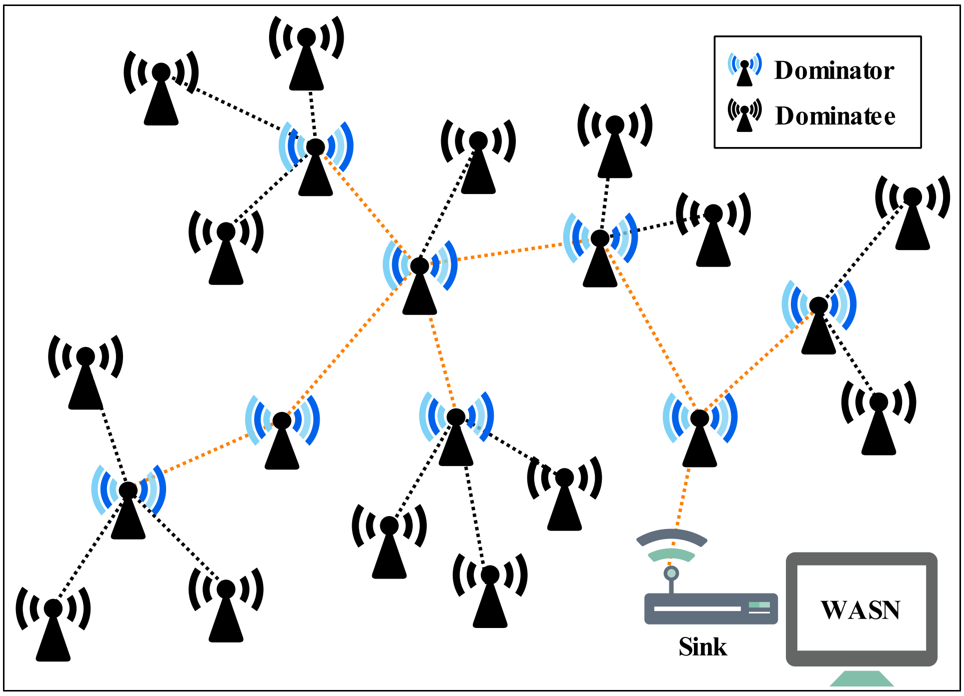

Figure 1.1 An example of WASN.

A capacitated MIS (CapMIS), capacitated DS (CapDS), and capacitated CDS (CapCDS) problems are extensions of MIS, DS, and CDS problems, respectively. If a node is in CapMIS, CapDS or CapCDS, it is called a dominator. Otherwise, it is a dominatee. Each dominator node $u \in V$ has a capacity c_u that determines the number of nodes it may dominate for these capacitated domination problems. The capacity is said to be hard if the dominatees of a dominator node is certainly limited according to its capacity. Otherwise, it is soft. On the other hand, the capacity of a dominator can be uniform or non-uniform. If each node in the graph can have a different (resp. same) capacity it is called non-uniform (resp. uniform). Non-uniform capacity is very appropriate for heterogeneous networks where uniform capacity is useful for homogeneous networks. Some dominators can cover a large number of dominatees, and the residual energy of them are consumed inefficiently. Thus, designing distributed self-stabilizing CapMIS, CapDS, and CapCDS algorithms are significantly important for energy efficiency, fault tolerance, and load balancing in WASNs in order to enhance the network lifetime.

In this thesis, we propose distributed self-stabilizing algorithms for CapMIS, CapDS, and CapCDS problems. All algorithms are theoretically proved in terms of closure and convergence (Dolev, 2000). Then we tested practically their performance on testbeds with IRIS motes and on simulations with a discrete event simulator TOSSIM. Simple, connected, undirected, and randomly generated UDG topologies are used with various node counts and densities. Although the system randomly starts from an illegal state, the proposed algorithms construct always a CapMIS, CapDS or CapCDS when the system is stabilized. All proposed algorithms showed significant performance by providing energy efficiency, scalability, and load-balancing in order to prolong the network lifetime.

1.1 Contribution

General contribution of the thesis is as follows:

- We propose the first distributed self-stabilizing soft capacitated MIS algorithm. It stabilizes at most $\left(\frac{5n^2}{6} + 3n\right)$ moves under an unfair distributed scheduler where a move is changing of the local state in an atomic step. The proposed algorithm is called A_{CapMIS} and has seven rules which are in priority order. It is theoretically proved in terms of convergence and closure.

- We propose the first distributed self-stabilizing hard capacitated DS algorithm. It constructs a 6-approximation CapDS at most $\left(\frac{5n^2}{3}+6n\right)$ moves for UDGs under an unfair distributed scheduler where a move is changing of the local state in an atomic step. The proposed algorithm called A_{CapDS} consists of eight rules which are in priority order.

- We present the first distributed self-stabilizing hard capacitated CDS algorithm. We suppose that a CDS is constructed by a distributed self-stabilizing algorithm (Kamei et al., 2016) before. Our algorithm converts a CDS structure to CapCDS. It can easily be composed with a distributed self-stabilizing CDS algorithm by using a hierarchical collateral composition technique (Datta et al., 2013). The proposed algorithm is called A_{CapCDS} and consists of six rules which are in priority order. It stabilizes at most $\left(\frac{n^2}{3}+2n\right)$ moves under an unfair distributed scheduler.

To the best of our knowledge, all proposed algorithms in this thesis are the first distributed self-stabilizing algorithms for CapMIS, CapDS, and CapCDS problems in the literature.

1.2 Outline of the Thesis

This thesis is organized as follows:

- Chapter 2 is dedicated to the concept of self-stabilization which includes information about communication models, self-stabilizing algorithm design, schedulers, complexity measures, and composition techniques.

- Chapter 3 presents the capacitated domination problems that are CapMIS, CapDS, and CapCDS. We give formal definitions, examples, and related works of the problems.

- Chapters 4, 5, and 6 present the first distributed self-stabilizing algorithms for CapMIS, CapDS, and CapCDS problems. The theoretical analysis and performance evaluations of them are given in detail in terms of closure and convergence.

- Finally, Chapter 7 concludes all the results of this thesis, and it gives some remarks and directions for further research.

2. SELF-STABILIZATION

2.1 Introduction

Constructing efficient distributed systems is tremendously desirable due to it performs well in realistic systems. Providing reliability, availability, and maintainability is considerably important in distributed systems. Reliability directly affects both availability and maintainability. The distributed systems are exposed to various kinds of faults such as hardware malfunction, battery drain, and link failure. These faults can occur at any time. Thus, designing a reliable distributed system is significantly important to cope with these faults to maintain a properly working system.

The faults in distributed systems are classified into three categories in terms of localization in time (Tixeuil, 2009). These are transient, permanent, and intermittent faults. If faults temporarily strike the system, and then the system goes to execution which these faults no longer occur, it is called transient faults. If faults strike the system execution permanently, it is called permanent faults. The last one called intermittent faults strikes the system at any moment in the execution. Fault tolerance is the capability of a distributed system to maintain properly desirable services without any interruption regardless of faults. The occurrence of faults can cause the system to reach an arbitrary state. A self-stabilization is a non-masking approach which shows the occurrence of faults to the observer and firstly introduced by Dijkstra (1974). An algorithm is self-stabilizing if it automatically recovers the system to a legitimate state in a finite time despite it initially starts from an arbitrary state, and it stays so without any external intervention. Self-stabilizing algorithms typically run in the background and never terminate. This property supports an adaptive fault tolerance which is more suitable for WASNs. A self-stabilizing algorithm must satisfy closure and convergence properties (Dolev, 2000) which are used to prove theoretically it. These properties follow as:

- Convergence: The system reaches a safe configuration in a finite time regardless of starting from any arbitrary configuration if no further faults occur during the stabilization.

- Closure: Once the system reaches a safe configuration, it stays so without any external interventions.

2.2 Communication Models

Communication is the heart of the distributed systems in which nodes cooperate to accomplish common tasks. Each node can only communicate with other adjacent nodes called neighbors within a one-hop distance. There are mainly three types of communication model for distributed self-stabilizing systems (Dolev, 2000).

1. Shared-memory model: A node in the system can read the local variables of its neighbors, but it can only change its variables in an atomic step.

2. Read/write atomicity model: A node can either read the local variables of its neighbors or update its local variable in an atomic step. Although the design is easier in the shared-memory model, this model is more realistic.

3. Message passing model: In this model, the neighboring nodes communicate with each other by sending and receiving messages which contain their local variables. A node can either send or receive a message in an atomic step.

Message passing model is more complicated than the other two models due to delay and message corruption by communicating. But it is more realistic for wireless networks. Moreover, there are two types of message passing model as synchronous and asynchronous.

1. Synchronous model: In the synchronous model, each node has a local clock which is exactly in sync with each other and execute in a lock-step called round. Messages are communicated in rounds. Each node can send only one message, and it is taken by all its neighbors in a round. It is not practical since realistic applications are subjected to various kinds of failures.

2. Asynchronous model: A system is asynchronous if there is no fixed upper bound on the message transmission delay between nodes. It is assumed that the messages are eventually delivered after unknown delays. Internet is a good example of an asynchronous model. This type of model is more suitable for real word applications since it does not require any strong assumptions on time and message order.

2.3 Self-stabilizing Algorithm Design

A distributed system can be represented by an undirected graph $G(V, E)$ where V is the set of nodes and E is the set of edges. Two nodes u and v are neighbors if and only if $(u, v) \in E$. N_v denotes the set of neighbors of node v. Formally, $N_v = \{u \in V | (u, v) \in E\}$. Each node has a set of local variables.

Definition 2.1 (State). *Each node v has a set of local variables called state. The state of node v is represented by $S_v \in P$, where P is all its possible states.*

Definition 2.2 (Configuration). *A configuration λ of a graph G consists of a tuple of all local states of the nodes $\lambda = (S_1, S_2, \dots, S_n)$. Γ denotes the set of all configurations.*

Definition 2.3 (Self-stabilization). *A system is self-stabilizing with respect to Λ such that $\Lambda \subset \Gamma$ if and only if it satisfies the convergence and closure properties.*

Definition 2.4 (Legitimate configuration). Any *configuration $\lambda \in \Lambda$ represents a legitimate configuration.*

Definition 2.5 (Execution). *An execution e of an algorithm is a maximal sequence of configuration $e = \{\lambda_1, \lambda_2, \dots, \lambda_i, \dots \lambda_m\}$ such that each configuration λ_{i+1} is the next configuration of λ_i in an atomic step.*

A self-stabilizing algorithm is generally written a collection of rule sets formed as “<label> ***if*** *<predicate>* ***then*** *<statement>*” in a priority order. A *predicate* is a Boolean expression that may be true or false depending on the values of its variables. If the *predicate* of any rule is satisfied, it is called *enabled*. A node is called *privileged* if at least one of its rules is enabled. If privileged node v executes the *statement* part of the rule, the local state S_v of node v is updated and this is called a *move*. If there is more than one enabled node of a privileged node, it can execute only one rule which has the highest priority. If all nodes in the system use the same program, it is called uniform. Otherwise, it is called a semi-uniform. If the nodes have a unique identifier in the system, this network is called *id*-based where an anonymous network has no identifiers. All algorithms discussed in this thesis are uniform and use an asynchronous message passing model on *id*-based networks.

2.4 Schedulers

The union of the local states of all nodes is called a global state, which can be either legitimate or illegitimate in distributed systems. There may exist more than one privileged node in any global state. A scheduler (daemon) is a virtual entity, which is assumed to select privileged nodes to decide which one will make a move. Only selected nodes make a move in an atomic step by a scheduler. In a distributed system, multiple selected privileged nodes can move simultaneously. Schedulers have an important role in calculating simply worst-case time complexity. Thus, the choice of a scheduler is tremendously important in designing of self-stabilizing algorithms. A good taxonomy for schedulers is presented by Dubois and Tixeuil (2011). The descriptions of the three schedulers are as follows:

1. Central scheduler: A central scheduler selects only one privileged node in an atomic step. Thus, it is not suitable for distributed systems.

2. Synchronous (Fully distributed) scheduler: A synchronous scheduler selects all privileged nodes in an atomic step. Therefore, it can be preferable for distributed homogeneous networks.

3. Distributed scheduler: A distributed scheduler selects a non-empty subset of the privileged nodes in an atomic step. This type of scheduler includes both central and synchronous scheduler. It is more suitable for realistic applications running on distributed heterogeneous networks.

It is easier to develop and analyze a distributed self-stabilizing algorithm running under a central scheduler, but the central scheduler is against the nature of the distributed systems since it forces the system control centralized and does not allow concurrent moves. Even if a synchronous scheduler allows the selected privileged nodes to move simultaneously, it causes a lot of overhead on the distributed systems by forcing synchrony. Moreover, it restricts the scalability of the network. Schedulers can be classified according to a fairness notion that represents the possibility of being selected node to make a move while being privileged. A (weakly) fair scheduler eventually selects a continuously privileged node while being privileged. An unfair scheduler may never select a continuously privileged node while there is at least one privileged node in the system except it. All algorithms proposed in this thesis run under an unfair distributed scheduler.

2.5 Complexity Measures

Complexity measures of self-stabilizing algorithms demonstrate the performance of them. The most common complexities are time complexity, space complexity, and message complexity. A node can read the local states of its neighbors, it checks the predicate of its rules, then it updates its local state by a move if it is privileged and selected by a scheduler in an atomic step in the asynchronous message passing, we used in this work.

Definition 2.6 (Move). *A local state transition of a node v from state S_v to S'_v after the execution of a privileged node selected by a scheduler is called a move.*

Definition 2.7 (Step). *A step is a tuple (λ, λ'), where λ' is the next configuration of λ after the privilege nodes in λ make a move simultaneously.*

Definition 2.8 (Round). *A round is a minimal sequence of steps in which each privileged node has a chance to move at the beginning of the round without affected by the move of its neighbors.*

Definition 2.9 (Time Complexity). *The time complexity of a self-stabilizing algorithm is the maximum number of moves, steps or rounds of the nodes from initially any arbitrary configuration to a legitimate configuration.*

The time complexity of a self-stabilizing algorithm can be measured in terms of the maximum number of rounds, the maximum number of steps or the maximum number of moves. Moves complexity has an upper bound of steps and rounds complexity, and there is generally a correlation formed $|round| \leq |step| \leq |move|$ between them. Besides, we can say that steps complexity is equivalent to moves complexity under a central scheduler, which selects one privileged node per step, and the rounds complexity is equivalent to steps complexity under a synchronous scheduler since a round contains only one step.

Most of the energy is consumed by the transceiver of a sensor node in order to send or receive a message in WASNs. A node broadcasts its current state to its neighbors when it moves. Reducing move count prolongs the lifetime of WASNs. Thus, move complexity is more suitable for self-stabilizing algorithms since it strongly demonstrates the efficiency of them.

2.6 Composition Techniques

Various composition techniques are often used to simplify the designing, analyzing, and proving the correctness of self-stabilizing algorithms (Tel, 2001). The most common use of them is collateral composition (Herman, 1992), fair composition (Dolev, 2000), conditional composition (Datta, 2000), and hierarchical collateral composition (Datta, 2013). Some complicated problems such as graph problems can be solved while composing at least two self-stabilizing algorithms. Suppose that A_1 and A_2 are two different self-stabilizing algorithms. Additionally, P_1 and P_2 are predicates over the variables of A_1 and A_2. The definitions of the composition techniques are below in detail.

Definition 2.10 (Collateral composition). *The collateral composition of* A_1 *and* A_2 *denoted* $A_2 o A_1$ *contains all local variables of* $A_1 \cup A_2$ *where* A_2 *read the variables of* A_1 *but* A_1 *does not. In collateral composition, both algorithms run concurrently where* A_2 *uses the output of* A_1*. When both algorithms are enabled at the same step, one or the other algorithm is executed nondeterministically.*

Definition 2.11 (Fair execution). *An execution* F_e *of* $A_2 o A_1$ *is fair according to* $A_i (i \in \{1,2\})$ *if one of the following conditions is valid:*

1. F_e *is finite.*

2. F_e *boundlessly contains many steps of* A_i.

Definition 2.12 (Fair composition). *The composition of* $A_2 o A_1$ *is fair according to* $A_i (i \in \{1,2\})$ *if any execution of* $A_2 o A_1$ *is fair according to* A_i. P_2 *will be constructed by* A_2 *after* P_1 *will be constructed by* A_1. $A_2 o A_1$ *stabilizes to* P_2, *if the following conditions are valid:*

1. A_1 *stabilizes to* P_1.

2. A_2 *stabilizes to* P_2.

3. *Once* P_1 *is true* A_1 *cannot change its variables while being read by* A_2.

4. *The composition is fair according to both* A_1 *and* A_2.

Definition 2.12 (Conditional composition). *A conditional composition of* A_1 *and* A_2 *represented* $A_2 o|_{Cond} A_1$. *It satisfies the following four conditions:*

1. $A_2 o|_{Cond} A_1$ *contains all local variables of* $A_1 \cup A_2$.

2. $Cond$ *is a subset of the predicates of* A_1.

3. *Each predicate* p_2 *of* A_2 *is formed as* $p_1 \wedge p_2$ *or* $\neg p_1 \wedge p_2$ *where* p_1 *is a Boolean expression using* $A_1 \in Cond$.

4. *If at least one rule of both* A_1 *and* A_2 *are enabled at the same step,* A_1 *moves after* A_2 *since* A_2 *uses the output of* A_1.

Definition 2.13 (Hierarchical Collateral Composition). *A hierarchical collateral composition is denoted by* $A_2 o A_1$ *and* A_2 *uses the outputs of* A_1. *It satisfies the following three conditions.*

1. $A_2 o A_1$ *contains all local variables of* $A_1 \cup A_2$.

2. $A_2 o A_1$ *contains all rules of* $A_1 \cup A_2$.

3. *Each rule* $P_i \rightarrow S_i$ *of* A_2 *is formed by* $\neg C \wedge P_i \rightarrow S_i$ *where* C *is the disjunction of all predicates of rules in* A_1.

In order to clarify the composition technique, we can give an example to solve a CDS problem. Suppose that A_1 is a self-stabilizing DS algorithm, and A_2 is a self-stabilizing Steiner tree algorithm. Moreover, P_1 and P_2 are predicates over the variables of A_1 and A_2. If the dominator nodes in DS constructed by A_1 is given A_2 as an input, we can construct a self-stabilizing CDS algorithm by using a composition technique. Suppose that we use hierarchical collateral composition technique. In this case, $A_2 o A_1$ has all variables of both A_1 and A_2. A_2 uses outputs (dominators in DS) of A_1 as inputs but vice versa is not true. The rules of A_2 is formed by $\neg P_1 \wedge P_2 \rightarrow S_2$ where S_2 is the statement of rules in A_2. This means that if A_1 has not an enabled rule in a step, A_2 can check its rules whether they are enabled or not. When A_1 is stabilized, A_2 can be in an arbitrary state. When both A_1 and A_2 are stabilized, a CDS is constructed. The time complexity of the composed self-stabilizing CDS algorithm is roughly measured by the sum of maximum rounds, steps or moves the complexity of A_1 and A_2.

3. CAPACITATED DOMINATION PROBLEMS

3.1 Capacitated Maximal Independent Set

3.1.1 Independent set problem

Clustering is a routing method that aims to block redundant data transmission and aims to optimize the usage of energy by forming a group of nodes called clusters. Each cluster has a cluster head (CH) which is responsible for the data aggregation, processing, and transmission. The other nodes in a cluster are called cluster member nodes (CMs) which collect and send the data to its CH. Generally, CHs consume more energy than CMs due to their relay task in the network. Hence, designing an efficient cluster head selection algorithm is significantly important for WASNs.

An independent set of an undirected graph $G(V, E)$ is a subset of nodes in which no two nodes have an edge of G where V is the set of nodes and E is the set of edges. If the size of nodes in IS is the largest, it is called a maximum independent set. Finding the maximum independent set problem is NP-hard. An IS is an MIS if it cannot be enlarged anymore. Designing an MIS algorithm for WASNs is very important, and it is widely used for clustering (Basagni, 2001, Alzoubi et al., 2003), routing, data aggregation, topology control, and building other graph structures. The nodes in MIS denote CHs, and the others denote CMs of clusters.

Define 3.1 (Independent Set). *An independent set (IS) for a given graph $G(V, E)$ is a subset $S \subseteq V$ such that there exists no node adjacent in S.*

Define 3.2 (Maximal Independent Set). *A maximal independent set (MIS) is an IS if no node can be added into IS without breaking independence. An MIS of maximum cardinality is called maximum.*

In Figure 3.1, an example of IS, MIS, and maximum IS is demonstrated. The black nodes denote the dominator nodes in IS, and the white nodes denote dominatee nodes out of IS. As illustrated in Figure 3.1.a, nodes 1 and 5 are in IS because they are not adjacent nodes. An MIS is shown in Figure 3.1.b since nodes 1 and 2 are in IS, they are not adjacent, and moreover the IS is not enlarged. Figure 3.1.c presents a maximum IS which has the maximum cardinality.

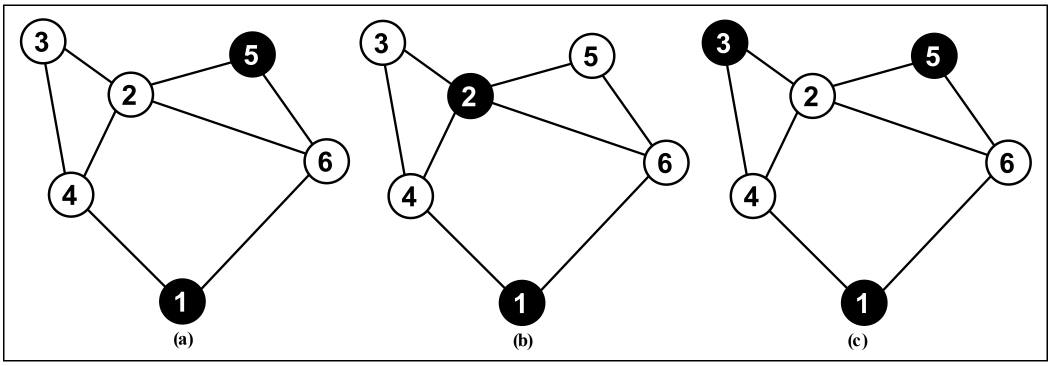

Figure 3.1 An example of a) IS b) MIS c) Maximum IS.

3.1.2 Capacitated maximal independent set problem

The capacitated MIS (CapMIS) problem is an extension of MIS in which each node $u \in V$ has a capacity c_u that determines the number of nodes it may dominate. The capacity is said to be hard if the dominatees of a dominator node in MIS is certainly limited according to its capacity. Otherwise, it is soft. Each node in (resp. out of) an MIS is denoted a dominator (resp. dominatee). Some dominators can cover a large number of dominatees, and the residual energy of them are consumed inefficiently. Thus, designing a CapMIS algorithm is significantly important for energy efficiency and load balancing in WASNs.

Definition 3.3 (Capacitated Maximal Independent Set). *A capacitated maximal independent set (CapMIS) of a graph G is an MIS such that each node in MIS has a capacity which is the number of nodes it may dominate.*

An example of CapMIS is demonstrated in Figure 3.2. The black nodes denote dominators in CapMIS, the white nodes denote dominatees out of CapMIS and have a dominator, and the grey nodes denote dominatees out of CapMIS and have a temporary dominator. The edge of the arrows shows the dominator node of a dominatee. In Figure 3.2.a, the capacity is non-uniform and equal to 3 and 2 for nodes 1 and 3 which are dominators, respectively. The dominator of nodes 2, 5, and 6 is 1. The dominator of nodes 4 and 7 is 3. This is a hard CapMIS since the capacity of the dominators does not overflow, and all dominatees has a dominator. As given in Figure 3.2.b, the capacity of nodes 1 and 3 is equal to 2. In this case, node 6 must select a temporary dominator which is either 1 or 3 since the capacity of nodes 1 and 3 are full if the selection policy of nodes 1 and 6 is a priority of minimum degree. This is a soft CapMIS since at least one dominator node must temporarily dominate a dominatee node until it chooses a permanent dominator.

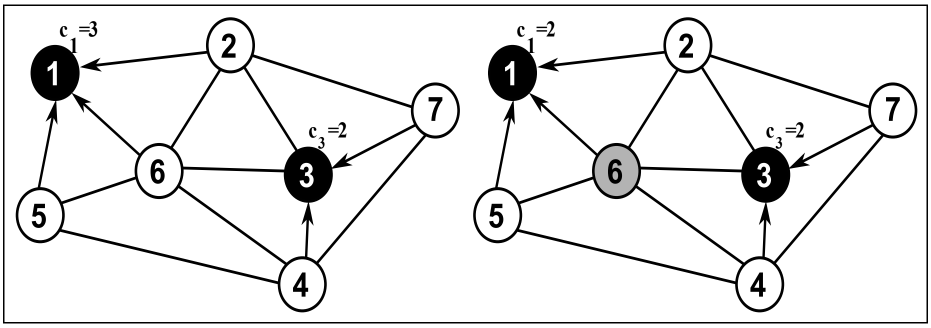

Figure 3.2 An example of a) Hard CapMIS b) Soft CapMIS.

3.1.3 Related work

The problem of computing an MIS has been studied by many researchers for decades because it is extensively used to solve many fundamental issues such as choosing CHs in clustering, capturing the essential challenge of symmetry breaking, building other graph structures. The algorithms are generally classified into two types which are central and distributed. The proposed central algorithms are more than distributed algorithms in the literature. Additionally, self-stabilizing central and distributed versions of MIS are very limited.

Karp and Wigderson (1985) gave a parallel MIS algorithm of which time complexity is $O((logn)^4)$ using $O((n/(logn))^3)$ processors in 1985. Alon et al. (1986) described a simple randomized MIS algorithm of which expected running time is $O(logn)$ with $O(|E|d_{max})$ processors where d_{max} is the maximum degree in the graph. Luby (1986) proposed two simple parallel MIS algorithms based on Monte Carlo algorithms. Goldberg and Spencer (1987) presented the first deterministic parallel algorithm for the MIS problem with $O(log^4n)$ running time. A parallel randomized algorithm for finding MIS in linear hypergraphs is presented by Luczak and Szymanska (1997). Blelloch et al. (2012) showed that the dependence length of the sequential greedy MIS algorithm is polylogarithmic $O(log^2n)$ with a high probability for any graph. Fischer and Noever (2018) proved a high probability upper bound of $O(logn)$ on the round complexity of (Blelloch et al., 2012) in general graphs. These algorithms are not suitable for distributed networks such as WASNs since they are centralized.

Kuhn et al. (2005) proposed a deterministic distributed algorithm that computes an MIS on bounded growth graphs in $O\ (log\Delta . log^*n)$ time where n and Δ represent the number of nodes and the maximal degree of the graph, respectively. Schneider and Wattenhofer (2008) presented a distributed MIS algorithm of which time complexity is $O\ (log^{*}n)$ on growth-bounded graphs and this bound is tight proven by (Linial, 1992). A distributed MIS algorithm is proposed and it achieves the optimal efficiency of $O(log\ n)$ expected time in (Scott et al., 2013). Ghaffari (2017) presented a randomized distributed algorithm providing a near-optimal local complexity for MIS via all-to-all communication. These distributed MIS algorithms are not self-stabilizing and they cannot provide fault-tolerance for unreliable platforms such as WASNs.

Guellati and Kheddouci (2010) presented a considerable survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. The first self-stabilizing MIS algorithm was introduced by Shukla et al. (1995). Hedetniemi et al. (2003) proposed a self-stabilizing MIS algorithm as similar to a self-stabilizing algorithm proposed by (Shukla et al.,1995). Lin and Huang (2003) gave an MIS algorithm, and it is an improvement of (Shukla et al., 1995). Shi et al. (2004) proposed a self-stabilizing algorithm for the (1-MIS) problem. All these self-stabilizing algorithms work under a central scheduler.

A distributed self-stabilizing MIS algorithm was introduced by Ikeda et al. (2002), and it stabilizes at most $O(n^2)$ moves. Then, Goddard et al. (2003) proposed a distributed self-stabilizing MIS algorithm which stabilizes at most $O(n^2)$ moves in $O(n)$ rounds. Both algorithms run under a fully distributed scheduler. Turau (2007) described a distributed self-stabilizing MIS algorithm stabilizes at most $max\{3n-5, 2n\}$ moves under an unfair distributed scheduler. Recently, Arapoglu et al. (2019) designed a distributed self-stabilizing MIS algorithm, and it stabilizes at most $max\{3n-6, 2n-1\}$ moves under a fully distributed scheduler.

Mentioned works do not consider on capacitated MIS problem, even if there are slightly distributed or centralized algorithms (not self-stabilizing) with soft or hard capacity for dominating set (Kuhn and Moscibroda, 2010; Shang and Whang, 2011; Kao et al., 2011; Cygan et al., 2011; Pradhan, 2012; Potluri and Singh, 2013; Liedloff et al., 2014; Kao et al., 2015; Li et al., 2017) and vertex cover (Chuzhoy and Naor, 2002; Guha et al., 2003; Gandhi et al., 2006; Kao et al., 2019) problems which are closely related with MIS problem in the literature. Designing a self-stabilizing CapMIS algorithm is rather difficult since no node in CapMIS is adjacent. To the best of our knowledge, there is no self-stabilizing CapMIS algorithm in the literature. Thus, we think that the proposed algorithm in this thesis is the first distributed self-stabilizing CapMIS algorithm.

3.2 Capacitated Dominating Set

3.2.1 Dominating set problem

Energy-efficient and fault-tolerant construction of dominating sets (DSs) on WASNs is one of the vital tasks which provide clustering, data aggregation, topology control, and routing. A DS is a subset of nodes in a graph $G(E, V)$ such that each node in DS is either a member of DS or a neighbor to a member of DS. It is a popular structure for WASNs (Thai and Du, 2006; Yang et al., 2012). Additionally, an MIS is a DS. The main difference between DS and MIS, dominator nodes can be adjacent in DS, but it cannot be allowed in MIS.

Definition 3.4 (Dominating set). A dominating set (DS) is a subset $S \subseteq V$ of a graph $G(V, E)$ in which each node $v \notin S$ is adjacent to at least one node $u \in S$. Formally, $S = \{v \in V: \forall\, t \in V - S: \exists\, u \in S: (u, t) \in E\}$.

Definition 3.5 (Minimal dominating set). A DS is minimal if it is not contained in any other DS of G. A DS is minimum if it has the smallest cardinality among all possible dominating sets of G. Finding minimum DS is NP-hard problem where minimal DS can be solved in polynomial time.

An example of CapDS is presented in Figure 3.3. The black nodes denote dominators in CapDS, the white nodes denote dominatees out of DS. In Figure 3.3.a, dominator nodes 1, 2, and 7 construct a minimal DS on a simple, connected and undirected graph since the other nodes have at least one dominator in their neighborhood. On the other side, Figure 3.3.b shows a minimum DS which is formed by nodes 4 and 7. In a graph, there may exist more than one minimum DS such that all of them has the same cardinality even if they contain different nodes.

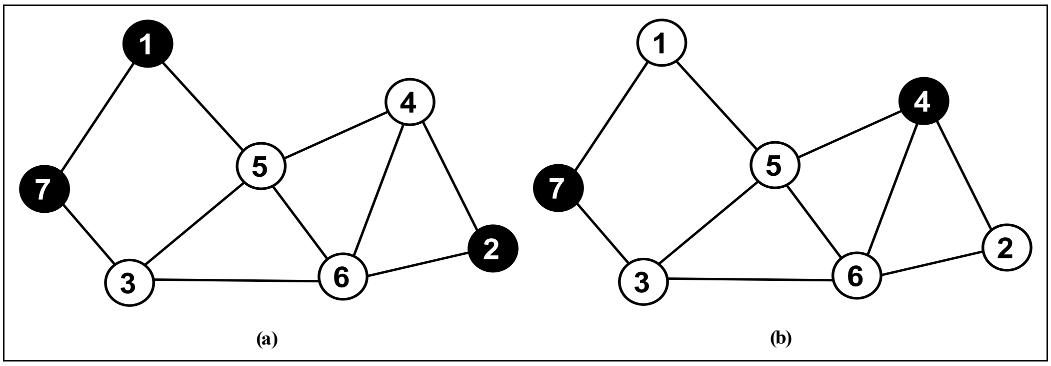

Figure 3.3 An example of a) Minimal DS b) Minimum DS.

3.2.2 Capacitated dominating set problem

A capacitated DS (CapDS) is a subset of nodes ($S \subseteq V$) where S is a DS, each non-dominator (dominatee) is assigned to a dominator, and a dominator cannot be matched with more than a predefined capacity (c) of dominatees. Obviously, the problem is to minimize set S, and it is NP-hard (Garey and Johnson, 1979). Design of a CapDS algorithm is especially a very important issue for energy-efficient clustering and load-balancing in WASNs where cluster sizes are bounded by a capacity value.

Definition 3.6 (Capacitated dominating set). *A capacitated dominating set CapDS of a graph $G(V, E)$ is a DS denoted by S such that each node in DS has a capacity which is the number of nodes it may dominate where V is the set of nodes and E is the set of edges. Let $V/S \rightarrow S$ be a mapping function which maps a dominatee node to a dominator. Formally, a CapDS* is defined *as* $S = \{v \in S: |\{u \in V - S: m(u) = v\}| \leq c\}$ *where c is a capacity.*

An example of CapDS is shown in Figure 3.4. Suppose that the capacity of each node is uniform and equal to 2. The black nodes denote dominators in CapDS and the white nodes denote dominatees out of CapDS. The edge of the arrows shows the dominator node of a dominatee. Otherwise, it has not chosen its dominator yet. Nodes 1, 5, and 8 constructs a CapDS. The dominator of node 3 and 7 is node 5, the dominator of nodes 2 and 6 is node 8, and the dominator of node 4 is node 1. Each dominatee has a dominator node, and the capacity of the dominator nodes does not overflow. Nodes 5 and 8 have full capacity except node 1. The system is stabilized since CapDS is constructed, and each neighbor of CapDS has exactly one dominator.

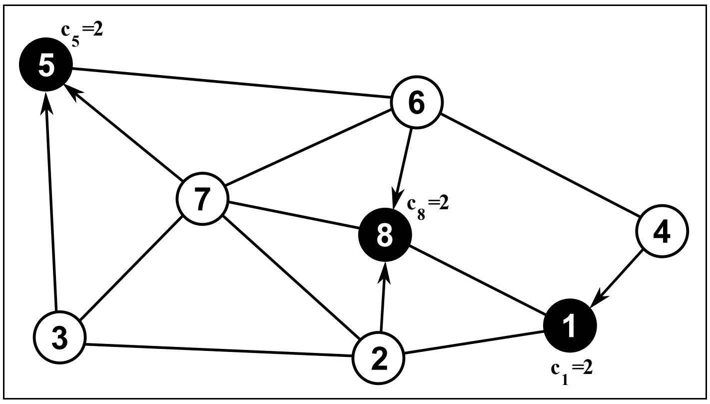

Figure 3.4 An example of CapDS.

3.2.3 Related work

Constructing dominating sets is a well-known problem has been studied by various researchers for decades. Kao and Liao (2007) described the (soft) capacitated domination problem with demand constraints such that it is to find a DS of minimum cardinality satisfying both the capacity and demand constraints. They presented a linear time $\frac{3}{2}$ – approximation algorithm for the unsplittable demand model and a pseudo-polynomial time algorithm for the splittable demand model. Dom et al. (2008) made an attempt to understand the behavior of the capacitated DS from the perspective of parameterized complexity. They showed that CapDS is $W[1] - hard$ when parameterized by both treewidth and solution size k of the CapDS. Then, Bodlaender et al. (2009) presented that planar CapDS which is the first bidimensional problem is $W[1] - hard$ by resolving an open problem of Dom et al. (2008).

Kuhn and Moscibroda (2010) proposed the first distributed algorithm for the minimum CapDS problem. This work became a pioneer for the distributed CapDS algorithms in the literature. Kao and Chen (2010) gave exact fixed-parameter tractable algorithms when parameterized by treewidth and the maximum capacity of the nodes. Shang and Wang (2011) proposed an approximation algorithm for the minimum CapDS problem. This central algorithm is a good starting point to understand clearly CapDS problem. Cygan et al. (2011) proposed an algorithm that solves CapDS exactly in $O(1{,}89^n)$ time. Potluri and Singh (2013) presented a heuristic algorithm and a couple of its variants for the minimum CapDS problem. They claimed that the heuristic algorithm works for both uniform and non-uniform capacity graphs. Additionally, it has better performance than its variants. Kao et al. (2015) presented a good survey on capacitated domination on problem complexity and approximation algorithms. Liedloff et al. (2014) solved minimum CapDS problem in $O^*(1.8463^n)$ time by using dynamic programming over subsets. Then, Backer (2016) proposed a polynomial-time approximation scheme for CapDS in unweighted planar graphs when the maximum capacity and maximum demand are bounded. A local search algorithm to solve CapDS is proposed by Li et al. (2017).

As mentioned above, there are less distributed algorithms for CapDS problem than centralized algorithms. However, studying on CapDS problem is going on from past to present. To the best of our knowledge, there is no a self-stabilizing central or distributed algorithm for CapDS problem.

3.3 Capacitated Connected Dominating Set

3.3.1 Connected dominating set problem

In WASNs, there is no fixed physical backbone. They contain sensor nodes which have limited power, memory, and computational capacities act as a router, mainly use a broadcast communication paradigm, and the data of the entire sensor network can be collected by a sink node. In order to cope with the scalability and energy efficiency of a WASN, it is necessary to construct a virtual backbone. A connected dominating set (CDS) is a DS which induces a connected subgraph of a graph $G(V, E)$ where V denotes the set of nodes and E denotes the set of edges. A virtual backbone of a WASN can be formed by a CDS (Kim et al., 2009). However, the construction of a minimum CDS is NP-hard (Clark et al., 1990) where a minimal CDS problem is solvable in polynomial time. Each node in (resp. out of) the CDS is called dominator (dominatee).

Definition 3.7 (Connected Dominating Set). A connected dominating set (CDS) of a graph $G(V, E)$ is a DS $S \subseteq V$, which induces a connected subgraph $G(S)$. A CDS is minimal if any proper subset of minimal CDS is not a CDS. A CDS is minimum if it has the smallest cardinality among all possible CDSs of G.

An example of CDS is presented in Figure 3.5. The black nodes denote dominators in CDS, the white nodes denote dominatees out of CDS. In Figure 3.5.a, dominator nodes 2, 3, and 7 construct a minimal CDS on a simple, connected and undirected graph since the other nodes have at least one dominator in their neighborhood, and the dominators are connected. As shown in Figure 3.5.b, a minimum CDS is formed by nodes 3 and 7.

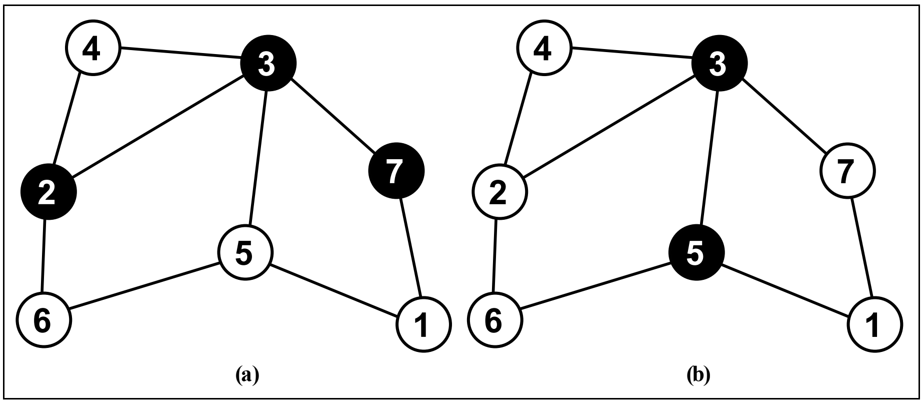

Figure 3.5 An example of a) Minimal CDS b) Minimum CDS.

3.3.2 Capacitated connected dominating set problem

As we mentioned above, a CDS is generally used to construct a virtual backbone in order to cope with the scalability and energy efficiency in WASNs. Although a CDS has an important role as a virtual backbone in WASN, some dominators can cover a large number of dominatees. Thus, it is obviously shown that there is no load balancing for a classical CDS problem. Alternatively, we can say a dominator v inefficiently consumes energy, another node u may not consume more energy since it has a few dominatees where v has a lot of dominatees given service by dominators. So, it is desirable to construct a CapCDS in order to provide load balancing, scalability, and energy efficiency which prolong the lifetime in WASNs.

Definition 3.8 (Capacitated connected dominating set). A capacitated connected dominating set CapCDS of a graph $G(V, E)$ is a CDS $S \subseteq V$ such that each node in S has a capacity c which denotes the number of nodes it can dominate where V is the set nodes and E is the set of edges of graph G.

In Figure 3.6, an example of CapCDS is presented. Suppose that the capacity of each node is non-uniform. A non-uniform capacity is more suitable for heterogeneous networks. The black nodes denote dominators in CapCDS, and the white nodes denote dominatees out of CapCDS. The edge of the arrows shows the dominator node of a dominatee. There are nine nodes which have a unique *id* from 1 to 9 increasing by 1. The capacity of 2, 7, and 8 are 3, 4, and 2, respectively. The dominator of nodes 5 and 9 is node 2, the dominator of nodes 3 and 6 is node 7, and the dominator of nodes 1 and 4 is node 8. The capacities of 2 and 7 are not full where the capacity of node 8 is full.

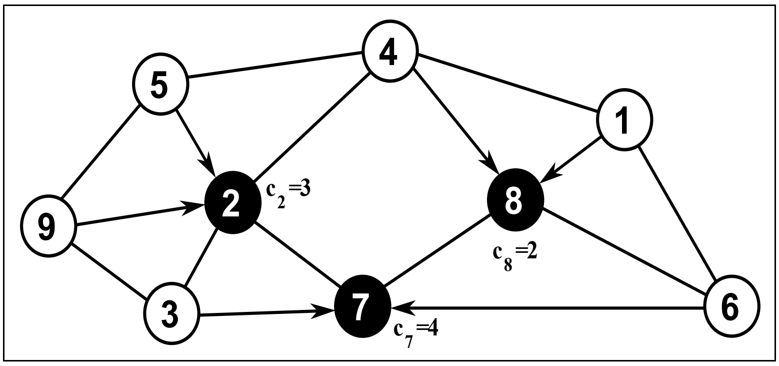

Figure 3.6 An example of CapCDS.

A dominatee node becomes a self-dominator when the capacity of all its neighbors are full. A self-dominator in CapCDS dominates only itself. This property of it is the difference between a dominator and a self-dominator. In Figure 3.7, an example of CapCDS with a self-dominator is shown. Nodes 1, 4, 6, and 8 construct a CapCDS but the only node 4 is a self-dominator since the capacity of node 1 is full. Node 2, 3, 5, 7, and 9 have a dominator, and the edge of the arrows shows their dominator.

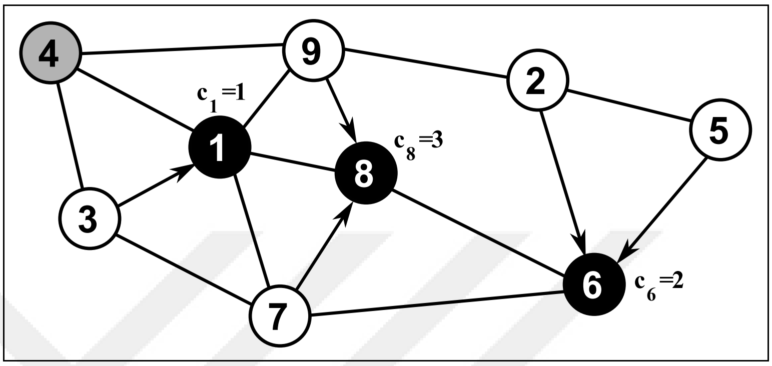

Figure 3.7 An example of CapCDS with a self-dominator.

3.3.3 Related work

Many researchers have studied on CDS problem since Ephremides et al. (Ephremides,1987) proposed the idea of using a CDS as a virtual backbone. CDS construction algorithms are generally classified into centralized and distributed. The literature has comprehensive surveys on CDS construction algorithms such as (Liu et al., 2010; Yu et al., 2013; Vijayasharmila, 2015; Vinayagam, 2016).

Guha and Khuller (1998), proposed two heuristic centralized CDS construction algorithms. Ruan et al. (2004) proposed a 1-phase greedy algorithm. The centralized algorithms are not suitable for decentralized networks such as WASNs. There exists many distributed algorithms such as (Wan et al., 2004; Gao et al., 2005; Funke et al., 2006; Cheng et al., 2006; Min et al., 2006; Raei et al., 2008; Gao and Zhang, 2012, Dhawan et al., 2014; Surendran and Vijayan, 2015; Jallu et al., 2017; Mohanty et al., 2017, Luo et al., 2018) in the literature. Mentioned works do not obviously consider fault tolerance and lack self-stabilization.

Jain and Gupta (2005) present a self-stabilizing distributed CDS algorithm which stabilizes a system at most $O(n^2)$ moves where n is the number of nodes.

Additionally, they assume that a node can read its neighbor information within a 3-hop distance and change variables within a 2-hop distance from it. This assumption is not efficient for self-stabilizing algorithms in WASNs. Then Kamei and Kakugawa (2007) proposed a self-stabilizing distributed approximation algorithm for finding the minimum CDS. The approximation ratio of the algorithm is at most $8|D_{opt}+1|$, and its time complexity is $O(n^2)$ moves for UDGs where D_{opt} is a minimum CDS. However, it has not a safe convergence property which is suitable for dynamic networks. They improved (Kamei et al., 2007) and gave a self-stabilizing distributed 7.6-approximation algorithm (Kamei et al., 2008) with safe convergence for the minimum CDS in UDGs. Unlike (Jain, 2005; Kamei et al., 2007; Kamei et al., 2008) work, Goddard and Srimani (2010) allow anonymous nodes and proposed two anonymous self-stabilizing distributed algorithms for CDS in a network graph. In (Kamei et al., 2012), Kamei and Kakugawa designed a self-stabilizing distributed 6-approximation minimum CDS algorithm with safe convergence in UDGs. They give a strong assumption for (Kamei et al., 2012) that each node executes the algorithm in the same step in parallel synchronously mentioned in (Herman, 2000). Then Kamei et al. (2016) proposed an asynchronous self-stabilizing distributed $(6+\varepsilon)$-approximation algorithm for the minimum CDS with safe convergence in UDGs. All these algorithms (Kamei et al., 2007; Kamei et al., 2008; Kamet et al., 2012; Kamei et al., 2016) are based on the strategy of Marathe et al.'s (1995) algorithm.

From the above discussions, it is clear that the mentioned works are not capacitated. Bar-Ilan et al. (1993) presented centralized approximation algorithms for NP-hard capacitated network center allocation problems. Shang and Wang (2011) proposed a centralized approximation algorithm for CapCDS problem. They suppose that a CDS is constructed by (Li et al., 2005) before. Their algorithm is heuristic, and the approximation ratio of it is $\left((k-1)/(k+1)\alpha+2\right)$ where $\alpha = 8 + ln5$ and k is the capacity that is uniform for all dominators. Khuller et al. (2014) study partial and budgeted versions of the CDS problem. They proposed two centralized approximation algorithms, and they obtain $O(lnq)$ and $\left(\frac{1}{3}\left(1-\frac{1}{e}\right)\right)$ −approximation ratio for the partial and budgeted versions of CDS problem where q is the quota for the partial version. They claim that they generalize their problems to CapCDS and weighted profit CDS which are a type of submodular optimization problems. To the best of our knowledge, there is no self-stabilizing CapCDS algorithm in the literature. In this thesis, we propose the first distributed self-stabilizing algorithm running under an unfair distributed scheduler for fault-tolerant minimal CapCDS construction in WASNs.

4. A DISTRIBUTED SELF-STABILIZING ALGORITHM FOR CAPACITATED MAXIMAL INDEPENDENT SET PROBLEM

4.1 Introduction

WASNs are composed of a large number of wireless self-organized sensor nodes connected through a wireless decentralized distributed network without the aid of a predefined infrastructure. Fault-tolerance and power management are fundamental challenges in WASNs. A WASN is self-stabilizing if it can initially start at any state and obtain a legitimate state in a finite time without any external intervention. Self-stabilization is an important method for providing fault-tolerance in WASNs. An MIS is extensively used for many important applications such as choosing cluster heads in clustering, building other graph structures in WASNs. The capacitated MIS (CapMIS) problem is an extension of MIS in which each node has a capacity that determines the number of nodes it may dominate. Designing a distributed CapMIS algorithm is significantly important in order to provide load balancing, scalability, and energy efficiency in WASNs.

In this section, we propose a distributed self-stabilizing capacitated maximal independent set algorithm. It stabilizes an unstable system at most $\left(\frac{5n^2}{6} + 3n\right)$ moves under an unfair distributed scheduler where n is the number of nodes, and move is a transition of the local states over a node in an atomic step. The proposed algorithm is theoretically proved in terms of convergence and closure properties of self-stabilization. Then, we test the performance of it on IRIS motes with testbeds and on TOSSIM with simulations. Simple, connected and undirected UDG topologies are used, and they are generated randomly for testbeds and simulations. Various node counts from 50 to 250 increasing by 50 in each step, and various network densities which are sparse, medium, and dense UDG topologies are used.

The remainder of this section is formed as follows. A system model is shown in Section 4.2. It demonstrates the predicates and the features of the environments in which the proposed algorithm runs and is tested. The design and analysis of the proposed algorithm are presented in Section 4.3. It explains the proposed algorithm in detail. In Section 4.4, the theoretical analysis of the proposed algorithm is given. The performance evaluations of the testbed experiments and the simulations are discussed in detail in Section 4.5.

4.2 System Model

This section launches by describing the system model of a WASN with distributed sensor nodes. We assume that all wireless ad hoc and sensor nodes are randomly deployed in a 2-D area, and they have a uniform transmission range. For simplicity, we model a WASN as a UDG $G(V, E)$ where V is the set of nodes and E is the set of edges. There exists an edge between any two nodes u and v if and only if the Euclidean distance between u and v is less than or equal to T_r. N_i denotes the neighbors of node i and id_i denotes the identifier of node i. In this work, we have made the following assumptions:

1. Each node has a distinct *id*.
2. The capacity of each node is non-uniform which is more suitable for real-world applications modeled heterogenous networks.
3. Communication links between nodes are bidirectional.
4. All nodes are homogeneously equipped except the sink node.
5. Each node knows its neighbors within its T_r.
6. The proposed algorithm is uniform since each node has the same program.
7. The rules of the proposed algorithm are executed atomically.
8. An unfair distributed scheduler is used as a runtime scheduler. Thus, it can select any non-empty subset of the privileged nodes at each step, and the selected nodes can execute their rules simultaneously.
9. Message passing model is used as a communication model. A node broadcasts its state if it moves, and the message is received by its neighbors within distance one-hop.
10. If the topology changes due to nodes joining or leaving the network, a new CapMIS should be constructed since the algorithm is self-stabilizing.

4.3 Proposed Algorithm

In this section, we present a distributed self-stabilizing algorithm for capacitated MIS problem under an unfair distributed scheduler. The proposed algorithm (A_{CapMIS}) is shown in Algorithm 4.1 and has seven rules. The rules are executed atomically in each step, and they are in priority order. A_{CapMIS} uses two states, and S_i denotes the state of node i. If a node i is in MIS, S_i is equal to IN. Otherwise, i is out of MIS and $S_i = \text{OUT}$. IN (resp. OUT) nodes are referred to as dominator (resp. dominatee). Each dominator node i has a capacity called c_i. A_{CapMIS} supports non-uniform capacity. So, c_i is a variable. Each node has a unique id, and id_i denotes the identifier of node i. N_i denotes the neighbors of node i. Each dominatee has $Dominator_i$ and $TempDominator_i$ variables which represent the dominator of node i, and the temporary dominator of node i, respectively. On the other hand, each dominator i has a set of dominatees denoted $Dominatees_i$.

In order to formally define the rules of A_{CapMIS}, the macros shown in Algorithm 4.1 are needed. $EmptyCapacity_i$ represents the empty space of $Dominatees_i$ to fulfill its capacity. $MaxEmptyNbr_i$ represents a neighbor IN node j which has the maximum size of $EmptyCapacity_j$. $CanDominatees_j$ represents the candidate dominatees which have OUT state, have not a dominator, and is not in $Dominatees_j$. $CanDominators_i$ represents the candidate dominators which have IN state, include node i in $Dominatees_j$, and $EmptyCapacity_j$ is greater or equal than zero. $MinNbr_i$ (resp. $MaxNbr_i$) denotes the neighbor node which has the minimum (resp. maximum) *id* in N_i. If a dominatee has at least an IN neighbor, $InNbr_i$ will be true. $MinInNbr_i$ shows the IN neighbor of node i which has the minimum *id*. $InNbrLower_i$ is true if there is at least one IN neighbor of i which *id* is lower than node i. The null value is shown $\perp$. The symmetry is broken by minimum *id* for all local variables if it exists.

A_{CapMIS} has 7 rules (Rs) where R1, R2, R3, and R4 are executed by IN nodes (dominators), and the other rules are executed by OUT nodes (dominatees). R1 and R7 are used to construct an MIS, and the other rules are given to provide the capacity constraint. If multiple rules are enabled in any configuration, the node executes the rule which has the highest rule number priority. An unfair distributed scheduler selects a non-empty subset of the privileged nodes in each step. The explanations of the rules are as follows:

Rule 1 (R1): If the state of node i is IN, and i has at least an IN neighbor of which id is lower than i, it changes state to OUT, sets both $Dominator_i$ and $TempDominator_i$ variables to null. This rule supports that any two IN nodes cannot be adjacent to construct an MIS.

Rule 2 (R2): In the initial state, the capacity of an IN node i can be overflow. In this situation, R2 is enabled. If i executes R2, it excludes the dominatees from $Dominatees_i$ until the capacity is not overflowed according to $MaxNbr_i$. There is no rule to cause that the capacity is overflow again by R2.

Rule 3 (R3): If the state of node i is IN, the size of $Dominatees_i$ is lower than c_i, and $CanDominatees_i$ is not empty, i adds the dominatees into $Dominatees_i$ until c_i is equal to the size of $Dominatees_i$ according to $MinNbr_i$ value of the dominatees. The first phase of a domination matching between a dominator and a dominatee starts by R3.

Rule 4 (R4): If a dominator node i with S_i = IN has at least a dominatee node j in $Dominatees_i$ which is not in N_i or $Dominator_j$ is not equal to i and not equal to $\perp$ or of which S_j is IN, i executes R4 to exclude j nodes from $Dominatees_i$. A wrong domination match is prevented by R4.

Rule 5 (R5): A dominatee node i choose its dominator if $Dominator_i$ is null, and there is at least one candidate dominator j which includes i in $Dominatees_j$. R5 completes the second phase of a domination matching after R3. If there is no candidate dominator in $CanDominators_i$, and $TempDominator_i$ is null, i chooses a temporary dominator j from IN neighbors and sets $TempDominator_i = j$ according to $MinInNbr_i$. If $EmptyCapacity_j$ of any IN neighbor node j is not null, it executes R3 and adds i into $Dominatees_j$ sooner or later. In this situation, i chooses j as a dominator and sets $TempDominator_i$ as null. Otherwise, $Dominator_i$ is null, and $TempDominator_i$ is not null. Thus, R5 supports a dominatee has a dominator permanently or temporarily when the system is stable by R5.

Rule 6 (R6): If a dominatee node i has a dominator j of which S_j is OUT or $Dominatees_j$ or N_j does not include i, i sets $Dominator_i = \perp$ and $TempDominator_i = \perp$. If $TempDominator_i$ is not null, $Dominator_i$ is not null or $TempDominator_i$ is not an element of N_i or the state of $TempDominator_i$ is

OUT, i sets $TempDominator_i = \perp$. R6 prevents that a dominatee has not a wrong dominator or temporary dominator.

Rule 7 (R7): If the state of a dominatee node i is OUT, i has not any IN neighbors and all neighbors with lower *id* than id_i have a dominator or temporary dominator, i changes the state to IN, enter CapMIS, and set $Dominatees_i = \emptyset$. There is no rule to force i to execute R1 again because its neighbors cannot enter CapMIS because they have an IN neighbor i. R1 and R7 support the construction of CapMIS together.

An example execution of A_{CapMIS} on a simple, connected and undirected UDG is presented in Figure 4.1. The initial configuration of the system is presented in Figure 4.1.a, and all nodes are privileged in the initial configuration. Each node has a unique *id* and a non-uniform capacity variable. There are two states of the nodes such that the state of black nodes is IN, and the state of white nodes are OUT. The edge of the arrows shows the dominator of a dominatee. A_{CapMIS} runs in steps under an unfair distributed scheduler. For simplicity, we suppose that all privileged nodes are selected by the unfair distributed scheduler in the first two steps. The steps of A_{CapMIS} follow as:

Step 1. Nodes 1 and 4 execute R6 and set $Dominator$ variables as null. Node 2 executes R2 and exclude node 7 from $Dominatees_2$ to prevent the capacity overflow. Since node 3 has not an IN neighbor and has the least *id* in its neighborhood, node 3 executes R7 and enters CapMIS by changing its state to IN. Node 5 executes R5 and set $Dominator_5 = 2$. Node 6 executes R3 and adds 5 into $Dominatees_6$. Node 7 executes R1 and changes its state to OUT because it has an IN neighbor with lower *id*.

Step 2. Node 1 executes R5 and sets $Dominator_1$ to node 2. Node 3 adds node 4 into $Dominatees_3$ by R3. Node 4 executes R5 and sets $TempDominator_4 = 3$. Node 6 executes R4 and excludes node 5 from $Dominatees_6$. Node 7 executes R5 and sets $Dominator_7 = 6$ since it is in $Dominatees_6$.

Step 3. In the third step, node 4 executes R5 and sets $Dominator_4 = 3$ and $TempDominator_4 = \perp$. The system is stabilized, and a CapMIS is constructed by nodes 2, 3, and 6 after this move. All dominatees with OUT state nodes have a dominator and all dominator nodes with IN state share the dominatees according to their capacity. Only dominator 2 has full capacity. The initial states of all nodes and convergence steps of A_{CapMIS} in Figure 4.1 are illustrated in Table 4.1. The stabilized system configuration is shown in Figure 4.1.b

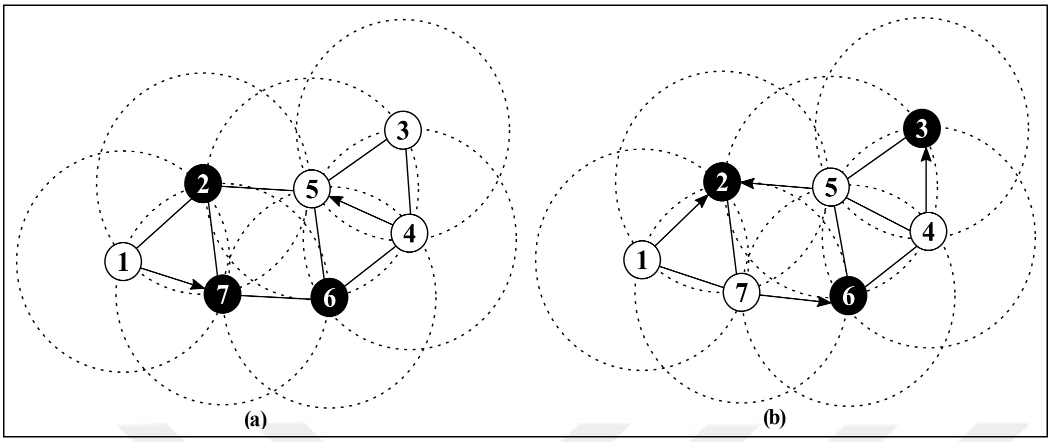

Figure 4.1 An example operation of A_{CapMIS} algorithm a) Initial state b) Stabilized state.

Table 4.1 Convergence steps of A_{CapMIS} in Figure 4.1.

	Initial States	**Step 1**	**Step 2**	**Step 3**
Node 1	$S_1 = OUT$ $c_1 = 2$ $Dominator_1 = 7$ $TempDominator_1 = \perp$	R6 $Dominator_1 = \perp$	R5 $Dominator_1 = 2$	
Node 2	$S_2 = IN$ $c_2 = 2$ $Dominatees_2 = \{1,5,7\}$	R2 $Dominatees_2 = \{1,5\}$		
Node 3	$S_3 = OUT$ $c_3 = 3$ $Dominator_3 = \perp$ $TempDominator_3 = \perp$	R7 $S_3 = IN$ $Dominatees_3 = \{\}$	R3 $Dominatees_3 = \{4\}$	
Node 4	$S_4 = OUT$ $c_4 = 2$ $Dominator_4 = 5$ $TempDominator_4 = \perp$	R6 $Dominator_4 = \perp$	R5 $TempDominator_4 = 3$	R5 $Dominator_4 = 3$ $TempDominator_4 = \perp$
Node 5	$S_5 = OUT$ $c_5 = 3$ $Dominator_5 = \perp$ $TempDominator_5 = \perp$	R5 $Dominator_5 = 2$		
Node 6	$S_6 = IN$ $c_6 = 4$ $Dominatees_6 = \{7\}$	R3 $Dominatees_6 = \{5,7\}$	R4 $Dominatees_6 = \{7\}$	
Node 7	$S_7 = IN$ $c_7 = 1$ $Dominatees_7 = \{\}$	R1 $S_7 = OUT$ $Dominator_7 = \perp$ $TempDominator_7 = \perp$	R5 $Dominator_7 = 6$	

Algorithm 4.1 A_{CapMIS}

Inputs.

id_i: *The identifier of node i.*

N_i: *The neighbors of node i.*

c_i: *The capacity of node i.*

Variables.

S_i: *The state of node i.*

$Dominator_i$: *The dominator of node i.*

$TempDominator_i$: *The temporary dominator of node i.*

$Dominatees_i$: *The dominatees set of node i.*

Macros.

$EmptyCapacity_i: \left|c_i - |Dominatees_i|\right|.$

$CanDominatees_i: \{j \in N_i | S_j = OUT \ \wedge Dominator_j = \perp \wedge\, j \notin Dominatees_i\}.$

$CanDominators_i: \{j \in N_i | S_j = IN \ \wedge i \in Dominatees_j \wedge EmptyCapacity_j \geq 0\}.$

$MaxEmptyNbr_i: j \in N_i | S_j = IN \ \wedge \ \forall t \in N_i(S_t = IN \ \wedge \ j \neq t \wedge EmptyCapacity_j \geq EmptyCapacity_t).$

$MinNbr_i: \min\{j \in N_i\}$, $MaxNbr_i: \max\{j \in N_i\}$.

$InNbr_i: \exists j \in N_i | S_j = IN$, $MinInNbr_i: min\{j \in N_i | S_j = IN\}$.

$InNbrLower_i: \exists j \in N_i | S_j = IN \wedge j < i.$

Rules.

R1. if $S_i = IN \wedge InNbrLower_i$ **then**

$S_i = OUT, Dominator_i = \perp, TempDominator_i = \perp$

R2. if $S_i = IN \wedge |Dominatees_i| > c_i$ **then**

repeat

$Pick\ MaxNbr_i \in Dominatees_i$

$Dominatees_i \coloneqq Dominatees_i \setminus \{MaxNbr_i\}$

until $S_i \neq IN \vee |Dominatees_i| \leq c_i$

R3. if $S_i = IN \wedge\ EmptyCapacity_i > 0 \wedge CanDominatees_i \neq \emptyset$ **then**

repeat

$Pick\ MinNbr_i \in CanDominatees_i$

$Dominatees_i \coloneqq Dominatees_i \cup \{MinNbr_i\}$

$CanDominatees_i \coloneqq CanDominatees_i \setminus \{MinNbr_i\}$

until $S_i \neq IN \vee\ EmptyCapacity_i = 0 \vee CanDominatees_i = \emptyset$

R4. 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

$Dominatees_i \coloneqq Dominatees_i \setminus \{j\}$

until $S_i \neq IN \vee\ \forall j \in Dominatees_i[(Dominator_j = i \vee Dominator_j = \perp) \wedge j \in N_i \wedge S_j \neq IN]$

R5. if $S_i = OUT \wedge\ Dominator_i = \perp \wedge [(CanDominators_i \neq \emptyset) \vee (TempDominator_i = \perp \wedge InNbr_i)]$ **then**

$\boldsymbol{if}\ CanDominators_i \neq \emptyset\ \boldsymbol{then}$

$Pick\ MaxEmptyNbr_i\ from\ CanDominators_i$

$Dominator_i \coloneqq MaxEmptyNbr_i, TempDominator_i \coloneqq \perp$

else

$TempDominator_i \coloneqq MinInNbr_i$

R6. if $S_i = OUT \wedge \left[\left(Dominator_i \neq \perp \wedge \left(i \notin Dominatees_{Dominator_i} \vee Dominator_i \notin N_i \vee S_{Dominator_i} = OUT\right)\right) \vee \left(TempDominator_i \neq \perp \wedge \left(Dominator_i \neq \perp \vee TempDominator_i \notin N_i \vee S_{TempDominator_i} = OUT\right)\right)\right]$ **then**

$TempDominator_i \coloneqq \perp$

if $Dominator_i \neq \perp \wedge \left(i \notin Dominatees_{Dominator_i} \vee Dominator_i \notin N_i \vee S_{Dominator_i} = OUT\right)$ $\boldsymbol{then}$

$Dominator_i \coloneqq \perp$

R7. if $S_i = OUT \wedge \neg InNbr_i \wedge \forall j \in N_i[i < j \vee Dominator_j \neq \perp \vee TempDominator_j \neq \perp]$ **then**

$S_i = IN, Dominatees_i = \emptyset$

4.4 Theoretical Analysis

4.4.1 Closure

Lemma 4.1 *When the system is stable,* $Dominator_i = j$ *if and only if* $i \in Dominatees_j$.

Proof. Assume, by contradiction, that the system is stable, and $Dominator_i = j$ but $i \notin Dominatees_j$. In this case, node i executes R6. If $i \in Dominatees_j$ and $Dominator_i \neq j$, it causes two cases. In case 1, if $Dominator_i = \perp$, node i executes R5. In case 2, if $Dominator_i \neq \perp$ and $Dominator_i \neq j$, node j executes R4. Since there is at least one move in a stable system, it is a contradiction.

Theorem 4.1 *In any state in which no node is enabled the set S is a CapMIS.*

Proof. Suppose to the contrary that the system is stable, and no node is enabled but S is not a CapMIS. Then either (*i*) S is not an MIS or (*ii*) S is an MIS but not capacitated. First consider (*i*), since S is not an MIS, there exists at least one node $i \notin S$ which has no IN neighbor. If node i has minimum *id* in its neighborhood or all of its OUT neighbors with lower *id* has a dominator or has a temporary dominator, R7 is enabled for i. If there exists at least an OUT neighbor node with a lower *id* which has not a dominator, R5 or R7 is enabled for it. This contradicts to the assumptions that no node is enabled. Now consider (ii), let node j is a node in S, and the capacity of j is overflow. In this situation, R2 is enabled. No rule lets the number of elements of $Dominatees_j$ be more than the capacity. Any node not in S must have at least one IN neighbor since S is an MIS. If node i does not have a dominator, and the capacity of node j which is the IN neighbor of node i is not full, then node j executes R3. If the capacity of all IN neighbors of i is full, i executes R5 and sets $TempDominator_i$ to $MinInNbr_i$. We contradict our assumption.

4.4.2 Convergence

Lemma 4.2 *A node can execute R2 at most once and as the first move.*

Proof. In the initial configuration, the number of $Dominatees_j$ of IN node j can overflow the capacity. In this case, R2 can be executed once as the first move, and the number of elements of $Dominatees_j$ can be at most equal to the capacity. No rule lets the number of elements of $Dominatees_j$ be more than the capacity. Thus, R2 can be executed at most once and as the first move.

Lemma 4.3 *A node executes R1 or R7 at most once. Only the (IN OUT IN) sequence and its suffix (OUT IN) are possible during the execution of A_{CapMIS} under an unfair distributed scheduler.*

Proof. In the initial configuration, suppose that an IN node i has an IN neighbor with a lower *id*, it must execute R1 as the first move. In order to execute R1 again, it must execute R7. If i has not an IN neighbor, i has minimum *id* in its neighborhood or the dominator or the temporary dominator of all neighbors with lower *id* are not null, i executes R7 and changes state to IN. After this move, its neighbors cannot execute R7. So, i remains in state IN so. At the end of these moves, i makes sequence IN OUT IN. Consequently, a node executes R1 or R7 at most once.

Lemma 4.4 *R5 and R6 can be executed at most $\frac{n^2}{2}$ times until the system is stable.*

Proof. Suppose that node i has initially state OUT, and it has at least one IN neighbor, it can execute R5 or R6. It causes two cases. In case 1, if $Dominator_i$ is not null, and i is not in dominatees set of $Dominator_i$, it executes R6 as the first move and sets $Dominator_i$ as null. After the first move, if i has an IN neighbor (j) which includes i in $Dominatees_j$, i executes R5. Node j can execute R1 and change state OUT. In this case, i executes R6. These moves (R6-R5) can repeat as long as i has IN neighbors which include i into their $Dominatees$ set and make (IN-OUT) move. There must be an IN neighbor (k) which has the minimum *id* among its IN neighbors and remains as IN. If k adds i into $Dominatees_k$, and i chooses k as dominator, i cannot execute any rule because there is no rule which breaks the matched of i and k. If k does not add i, an OUT neighbor (u) which made (IN-OUT) move before executes R7 and adds i into $Dominatees_u$ by R3, u cannot execute R1 again by Lemma 4.3, and i chooses u as a dominator, i cannot execute any rule because there is no rule which breaks the matched of i and u.

In case 2, if $Dominator_i$ is null, $TempDominator_i$ is not null, and $TempDominator_i$ is initially not in N_i, i executes R6 and sets $TempDominator_i$ as null. If i has an IN neighbor (j), it chooses j as $TempDominator_i$ by executing

R5. j can execute R1 and change state OUT. Then, i executes R6 and sets $TempDominator_i$ as null. These moves (R6-R5) repeat as long as IN neighbors of i make (IN-OUT) move. If an OUT neighbor (u) which made (IN-OUT) move before executes R7, i chooses u as a $TempDominator_i$ as long as $Dominator_i$ is null.

The two cases are mutual exclusion. The following formula shows the greatest move count of R5 and R6 for each case where n is the set of nodes of graph G:

$$= 2xy(x = \{S_i = OUT\}, y = \{S_j = IN\}, n = x + y$$

$$= 2x(n - x)$$

$$= 2nx - 2x^2$$

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

Lemma 4.5 *R3 and R4 can be executed at most* $\frac{n^2}{3}$ *times until the system is stable.*

Proof. In the initial configuration, the system may not be stable. Suppose that the state of node i is initially IN state, it can make IN OUT IN sequence. In order to make this sequence, it must execute IN OUT sequence as the first move by Lemma 4.3. Suppose that there are n nodes in the system. In the initial configuration, let X is the set of nodes in CapMIS, Y is the set of other nodes, $|X| = x$ and $|Y| = y$. Any node in Y has a neighbor of at least one node in CapMIS. In the first step, x nodes can execute R3, and all of them can add the same node into their $Dominatees$. In the second step, at least one node of Y executes R5. So, the capacity of every node in X must be one and equal in the worst-case scenario. In the third step $(x - 1)$ nodes can execute R4 and remove the matched node in Y from their $Dominatees$. In the fourth step, $(x - 1)$ nodes can execute R3 and add the same node into their $Dominatees$ from Y except the matched node. In the fifth step, at least one node of Y executes R5 and chooses a dominator from X. It is shown below with equations that how many times the dominators execute R3 and R4 totally until the system is stable.

$$x + y = n(x \geq 1, y \geq 1, n \geq 2)$$

$$x + (x-1) + (x-1) + (x-2) + (x-2) + \cdots$$

$$= \begin{cases} x + 2\sum_{i=1}^{x-1}(x-i), & x \le y \ (4.1) \\ x + 2\sum_{i=1}^{y-1}(x-i) + x - y, & x > y \ (4.2) \end{cases}$$

$Case\ 1{:}\ if\ x \le y$

$$x + 2\sum_{i=1}^{x-1}(x-i)$$

$$= x + 2\left(\sum_{i=1}^{x-1} x - \sum_{i=1}^{x-1} i\right)$$

$$= x + 2\left((x(x-1) - \frac{x(x-1)}{2}\right)$$

$$= x + 2(x^2 - x) - x^2 + x$$

$$= 2x - 2x + 2x^2 - x^2$$

$$= x^2$$

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

$Case\ 2{:}\ if\ x > y$

$$x + 2\sum_{i=1}^{y-1}(x-i) + x - y$$

$$= x + 2\left(\sum_{i=1}^{y-1} x - \sum_{i=1}^{y-1} i\right) + x - y$$

$$= 2x - y + 2\left(x(y-1) - \frac{y(y-1)}{2}\right)$$

$$= 2x - y + 2(xy - x) - y^2 + y$$

$$= 2xy - y^2$$

$$= 2(n - y)y - y^2$$

$$f(y) = 2ny - 3y^2$$

$$f(y)' = 2n - 6y = 0$$

$$y_{max} = \frac{n}{3} \; and \; \frac{n^2}{3} \; is\ the\ greatest\ move\ count.$$

$$Since \; \frac{n^2}{4} < \frac{n^2}{3} \; for\ n \geq 0, so\ until\ being\ stable\ R3\ and\ R4\ can\ be$$

$$executed\ at\ most \; \frac{n^2}{3} \; times.$$

Theorem 4.2 *A_{CapMIS} is self-stabilizing under an unfair distributed scheduler and stabilizes after at most $\left(\frac{5n^2}{6} + 3n\right)$ moves with a CapMIS.*

Proof. Any node can execute R1 or R7 at most once by Lemma 4.3. It causes at most $2n$ moves. From Lemma 4.2, any node executes R2 at most once. Thus, n moves can be executed totally at most for R2. Any node can execute R5 and R6 at most $\frac{n^2}{2}$ times by Lemma 4.4. R3 and R4 can be executed at most $\frac{n^2}{3}$ times by nodes from Lemma 4.5. Therefore, the total move count in the worst-case scenario is bounded by:

$$= 2n + n + \frac{n^2}{2} + \frac{n^2}{3}$$

$$= \frac{5n^2}{6} + 3n$$

4.5 Performance Evaluation

4.5.1 Testbed experiments

The proposed algorithm is evaluated through IRIS motes based on the ATmega1281 microcontroller that increase from 10 to 40 step by 10 in the testbed. IRIS motes have 2.4 GHz IEEE 802.15.4 compliant transceiver, 250 kbps data rate, 8 kB RAM, 128 kB programmable flash memory. A_{CapMIS} is written in NesC language supported by TinyOS and tested on TOSSIM with UDGs. Simple, connected and undirected UDG topologies are randomly generated. Each UDG topology consists of sensor nodes which have equal-sized T_r and are deployed randomly. The topologies are classified in three densities which are sparse, medium, and dense where average degrees of these topologies are 4, 6, and 8, respectively. Java-based gateway software is developed to listen to the motes in the testbed via a sink node connected to a notebook.

We measured the move count, transmitted byte count, received byte count and energy consumption (mJ) of A_{CapMIS} against various node counts and densities. A carrier sense multiple access with collision avoidance MAC protocol is used in order to reduce the packet interference probability. The state of a node can be OUT or IN. The dominators send id_i, S_i, c_i, and $Dominatees_i$ variables where the dominatees send id_i, S_i, and $Dominator_i$ variables in a message packet if they move. The local variables of each node are initially started randomly since a self-stabilizing system can be started from any initial configuration. When the system is stabilized, a CapMIS which includes only OUT and IN nodes is created.

A non-uniform capacity is used for testbeds and simulations since it is very appropriate for heterogeneous networks which are more realistic. The maximum data bytes sending in a message packet do not exceed 127 bytes in the structure of IEEE 802.15.4. E_{max} denotes the maximum consumed energy for sending a 127 byte-sized packet, and E_i denotes the energy of node i. Initially, we set random energy to each node of which energy is varying between $1000xE_{max}$ and $10000xE_{max}$. D_i denotes the degree of node i, and c_i denotes the capacity of node i. E_{avg} and D_{avg} denote the average energy and degree of the neighbors, respectively. We calculate the initial capacity of the nodes considering their energy as follow:

$$c_i = \begin{cases} \left\lceil \dfrac{E_i D_{avg}}{E_{avg}} \right\rceil, & if \left\lceil \dfrac{E_i D_{avg}}{E_{avg}} \right\rceil \leq D_i \ (4.3) \\ D_i \,, & if \left\lceil \dfrac{E_i D_{avg}}{E_{avg}} \right\rceil > D_i \ (4.4) \end{cases}$$

Move count is a fundamental criterion affecting energy consumption due to decreasing move count prolongs the network lifetime in WASNs. The move count of A_{CapMIS} against the node count and density is given in Figure 4.2.a. When the node count rises, the move count of A$_{CapMIS}$ rises linearly. Move counts are at least in sparse topologies.

Transmitted and received byte counts are directly affected by the move count. Because, when a node moves it transmits the new state of its local variables to its neighbors. Then, the transmitted message packets are received by its neighbors within one-hop distance. As illustrated in Figure 4.2.b and Figure 4.3.a, the transmitted and received byte counts are compatible with the move count shown in Figure 4.2.a. When the node count increases, the transmitted and received bytes increase linearly. If the density of the network increases, the transmitted and received bytes increase directly. However, the difference is more for the received byte count than the transmitted byte count since the node degree affects the received byte count directly. By these results, A$_{CapMIS}$ has high scalability due to act in response to rising node count and network density.

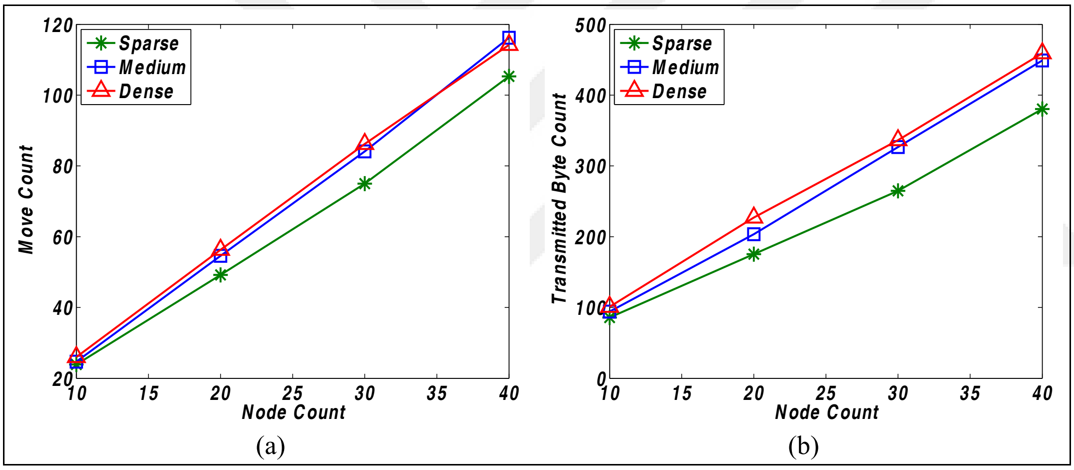

Figure 4.2 a) Move count b) Transmitted byte count of A$_{CapMIS}$ against node count and density.

The amount of energy consumption is calculated for each node from the transmitted bytes (T) and received bytes (R). Transmit data rate of IRIS motes is 250 kbps (i.e. 31.25 kBps). They consume approximately 16 mA current in the receive mode and 17 mA in the transmitted mode with TX = 3 dBm (MEMSIC, 2019). The input voltage of these motes is 3300 mV. According to the general formula E = V × I × T, the energy consumption is calculated as follow:

$$E \approx \left(\frac{17T + 16R}{32}\right) 3.3mJ \quad (4.5)$$

Energy consumption is the most important parameter affecting network lifetime in WASNs. Most of the energy is consumed for communication (Turau, 2007). Thus, the transmitted and received byte counts affects directly the energy consumption. As indicated in Figure 4.3.b, energy consumption rises linearly when the node count or the density rises. It is higher in dense topologies in which the transmitted messages of node i are received by all neighbors of node i.

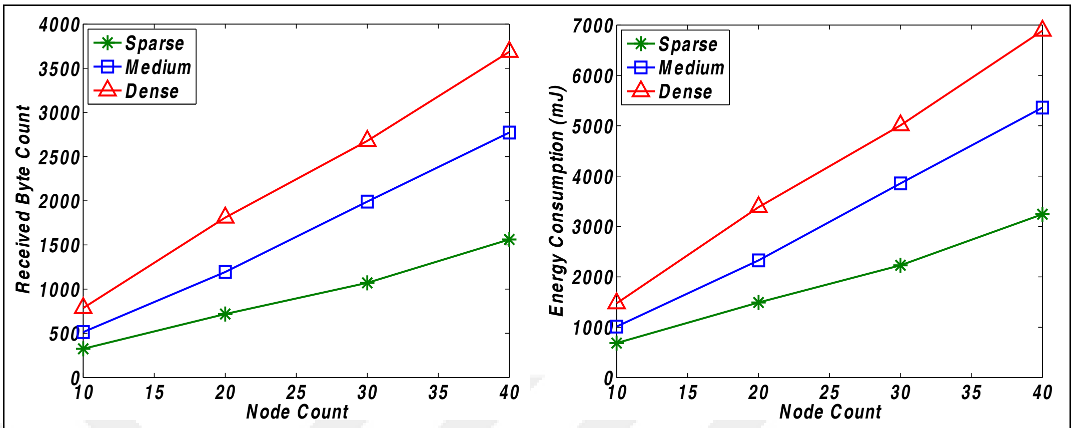

Figure 4.3 a) Received byte count b) Energy consumption of A_{CapMIS} against node count and density.

The results observed from the figures of the testbeds show that the proposed algorithm A_{CapMIS} is stable against various node counts and densities. Although the system is randomly started from any configuration, a CapMIS is always created when the system is stabilized without any external intervention. Moreover, A_{CapMIS} has high scalability and energy efficiency in WASNs.

4.5.2 Simulations

The performance of A_{CapMIS} is evaluated on a discrete event simulator TOSSIM for TinyOS sensor network under an unfair distributed scheduler. Simple, connected and undirected UDG topologies that are modeled of WASNs are randomly generated where the node count is varying from 50 to 250 by increasing 50 at each step. The densities of the networks are classified into three types which are sparse, medium, and dense of which average degrees are approximately 4, 6, and 8, respectively. Each UDG topology consists of sensor nodes which have equal-sized T_r which covers one-hop distance. However, the proposed algorithm cope with not different-sized T_r.

Each node is initially assigned with a unique *id* (id_i), non-uniform capacity (c_i), S_i, $Dominator_i$, $TempDominator_i$, and $Dominatees_i$ local variables. The capacities of the nodes are randomly generated with Formula 4.3 or 4.4. Initially, the dominators broadcast id_i, S_i, c_i, and $Dominatees_i$ variables where the dominatees broadcast id_i, S_i, and $Dominator_i$ variables in a *Hello* message packet. Then they send these packets if they are selected by an unfair distributed scheduler to move until the system is stabilized. In order to evaluate the performance of A_{CapMIS}, we measure move count, transmitted byte count, received byte count, energy consumption, and the lifetime of the networks. Each measurement is the average of 50 repeated simulations.

To the best of our knowledge, there is no distributed self-stabilizing CapMIS algorithm in the literature. Thus, we designed two distributed self-stabilizing CapMIS algorithms from (Arapoglu et al., 2019) which is the best energy efficient and distributed self-stabilizing MIS algorithm in the literature so far by applying a hierarchical collateral composition technique. The first algorithm called as A_{Random} has a random approach in which the dominatees choose randomly its dominator, and the dominators definitively add them into their dominatees set if the node does not execute any rule of (Arapoglu et al., 2019). The second algorithm called as A_{Greedy} uses a minimum *id* priority-based approach in which the dominatees always choose the dominator which has the minimum *id*, and the dominators add definitively them into their dominatees set if the node does not execute any rule of (Arapoglu et al., 2019). We implemented and tested A_{CapMIS}, A_{Random}, and A_{Greedy} on TOSSIM.

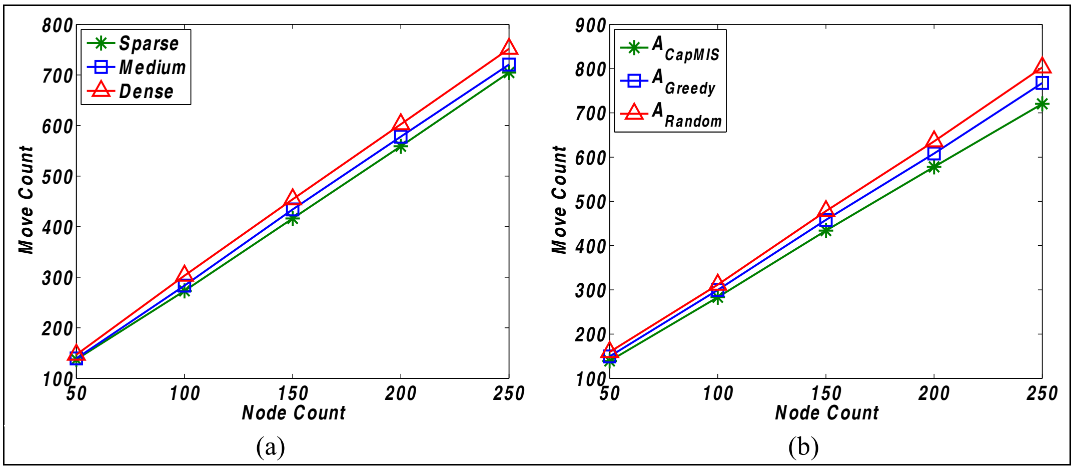

Figure 4.4 a) Move count of A_{CapMIS} against node count and density. b) Move count of algorithms with medium density against node count.

Move count is one of the most significant criteria affecting energy consumption since decreasing move count is vital to prolonging the network lifetime. As shown in Figure 4.4.a, the move count of A_{CapMIS} rises linearly when the node count rises. Sparse networks have at least move count. A comparison of the algorithms in terms of move count is represented in Figure 4.4.b. Move counts of the algorithms increase directly proportional with the node count where A_{CapMIS} is 1.10 times better than A_{Random} and 1.05 times better than A_{Greedy}. These results show us that the move counts of A_{CapMIS} are significantly smaller than its counterparts although increasing node counts.

As illustrated in Figure 4.5.a, the transmitted byte count increases when the node count increases. It is at least in the sparse graphs. In Figure 4.5.b, the transmitted byte count of the algorithms against node count and density is given. Although the behavior of the algorithms is similar against the varying node count and density, A_{CapMIS} is 1.14 times better than A_{Random} and 1.07 times better than A_{Greedy}. It is obviously shown that A_{CapMIS} has better performance in terms of transmitted byte count.

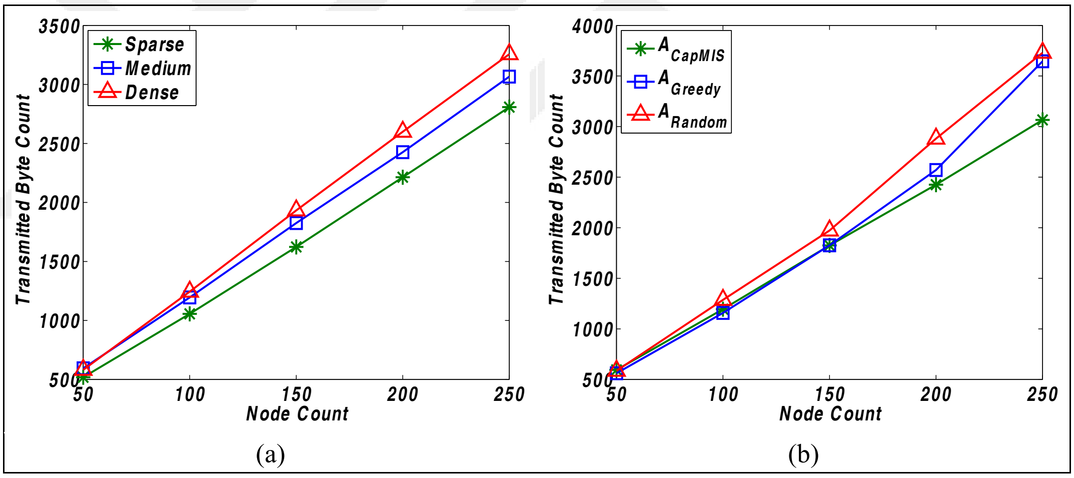

Figure 4.5 a) Transmitted byte count of A_{CapMIS} against node count and density. b) Transmitted byte count of algorithms with medium density against node count.

If a node moves, it transmits the new local variables to its neighbors in order to inform them. The transmitted message packets are taken by its neighbors. Thus, the transmitted byte count affects directly received byte count. As indicated in Figure 4.6.a, when the node count or density increases, the received byte count increases linearly. We can say that A_{CapMIS} is stable against various node counts and densities. A comparison of the algorithms in terms of the received byte count is shown in Figure 4.6.b. A_{CapMIS} is 1.11 times better than A_{Random} and 1.03 times better than A_{Greedy}. According to the received byte count, A_{CapMIS} has the best performance among the algorithms.

Energy consumption is directly affected by the transmitted and received byte counts. Since the nodes lack rechargeable battery, it has vital importance in WASNs. We calculated the energy consumption from Formula 4.5 since transmitting or receiving the message are the fundamental consumers in a message passing communication model in WASNs. In Figure 4.7.a, the energy consumption of A_{CapMIS} against node count for various densities is presented. When the node count is enlarged, the energy consumption increases in a natural way. The energy consumption of A_{CapMIS} increases linearly when the density increases. As illustrated in Figure 4.7.b, A_{CapMIS} is 1.11 times better than A_{Random} and 1.03 times better than A_{Greedy}. The results observed from the figures show that the proposed algorithm is significantly energy efficient than its counterparts.

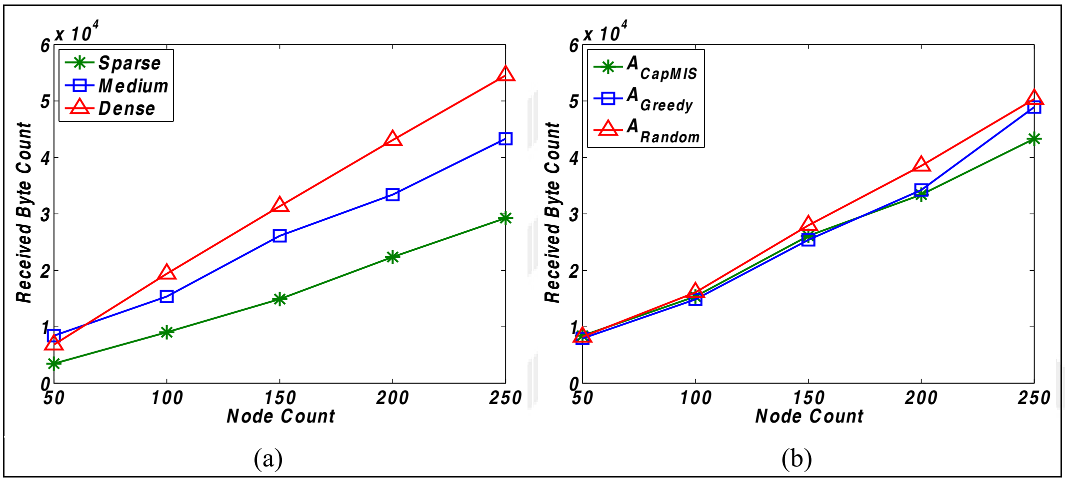

Figure 4.6 a) Received byte count of A_{CapMIS} against node count and density. b) Received byte count of algorithms with medium density against node count.

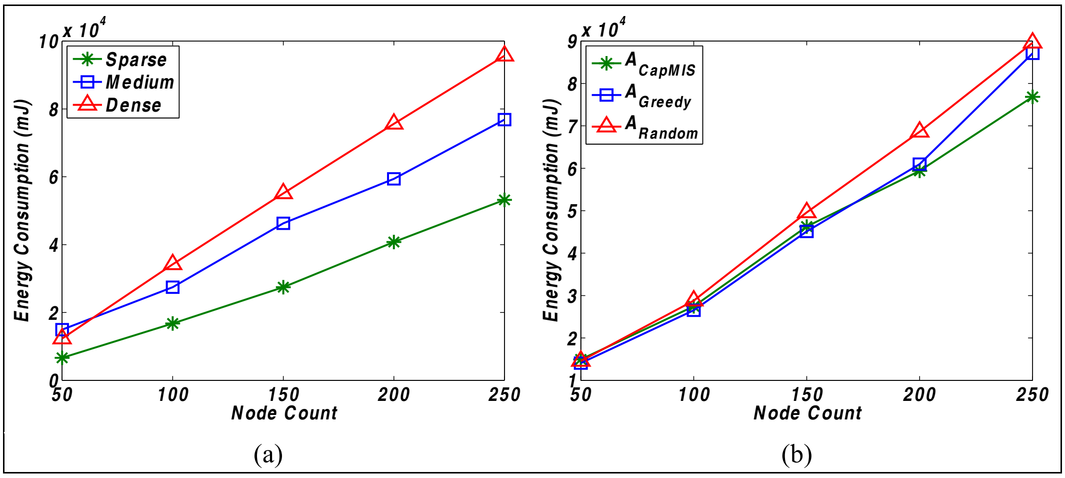

Figure 4.7 a) Energy consumption of A_{CapMIS} against node count and density. b) Energy consumption of algorithms with medium density against node count.

The network lifetime is a vital issue in WASNs. The lifetime of a WASN is widely defined as the time from the network starts execution to the first node failure. Dominators consumed undoubtedly more energy than the dominatees due to their CH role in a clustering. The lifetime of a dominator denoted L_i is calculated by Formula 4.6 as $\min\{L_i\}$. In Figure 4.8.a, the lifetime of A_{CapMIS} against node count and density is shown. In spite of the fact that the lifetime generally fluctuates, when the node count or density increases the lifetime decreases. Because there are fewer dominators in CapMIS in dense topologies due to the probability of a node with less energy became a dominator is low in the dense topologies. As shown in Figure 4.8.b, A_{CapMIS} has the best network lifetime where A_{CapMIS} is 2.06 times better than A_{Random} and 2.21 times better than A_{Greedy}.

$$L_i = \frac{E_i}{(E_{max})|Dominatees_i|} \qquad (4.6)$$

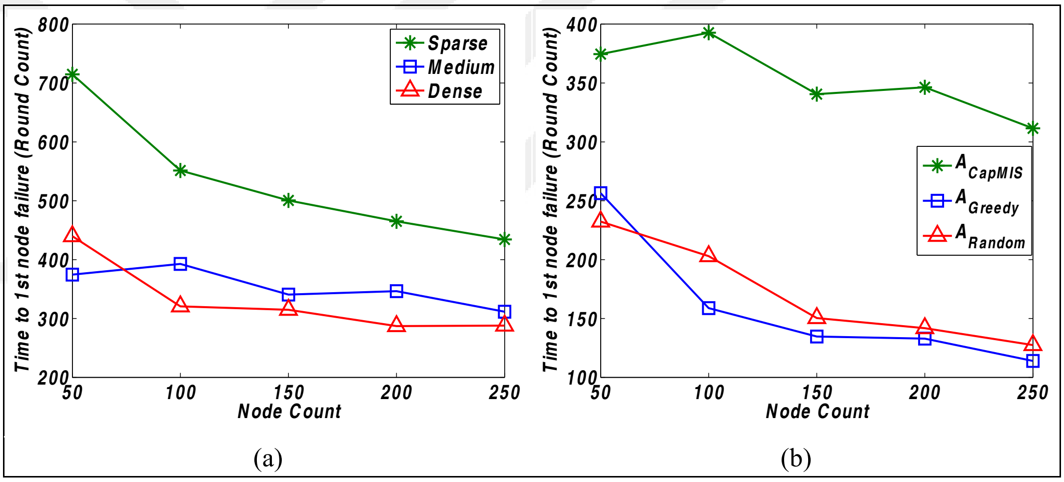

Figure 4.8 a) Lifetime of A_{CapMIS} against node count and density. b) Lifetime of algorithms with medium density against node count.

As a result, the simulation results are evidence that A_{CapMIS} has high stabilization and scalability despite large-scale networks and various network densities. The simulation results are compatible with the testbed results. Moreover, they proved that A_{CapMIS} copes with non-uniform capacities under an unfair distributed scheduler. When the system stabilized, each dominatee has a dominator or a temporary dominator and there are no adjacent nodes in CapMIS. A_{CapMIS} is compared with A_{Random} and A_{Greedy}, and it is more energy efficient than both of them since the distribution of dominatees according to the capacity of dominators is more balanced in A_{CapMIS}. Therefore, we can obviously say that A_{CapMIS} significantly prolongs the network lifetime in WASNs.

5. A DISTRIBUTED SELF-STABILIZING ALGORITHM FOR CAPACITATED DOMINATING SET PROBLEM

5.1 Introduction

Energy-efficient and fault-tolerant construction of dominating sets (DSs) on WASNs is one of the vital tasks which provides clustering, data aggregation, topology control, and routing. A WASN is self-stabilizing if it can initially begin at any state and obtain a legitimate state in a finite time without any external intervention. In this thesis, we propose a distributed fault-tolerant algorithm for a minimum capacitated DS (CapDS) construction in WASNs. To the best of our knowledge, this is the first self-stabilizing CapDS algorithm. We proved the self-stabilization and asynchronous behaviors of the algorithm in terms of closure and convergence. Moreover, we showed that the proposed algorithm has a 6-approximation ratio for WASNs modeled as UDGs. The proposed algorithm can run on all connected graphs which are id-based, but we can give a 6-approximation ratio for only UDGs.

The remainder of this section is formed as follows. In Section 5.2, the system model is presented. It demonstrates the predicates and the features of the environments in which the proposed algorithm runs and is tested with testbeds and simulations. The design and analysis of the proposed algorithm are given in Section 4.3. It presents the explanation of the proposed algorithm in detail. The theoretical analysis of the proposed algorithm is demonstrated in Section 4.4. The performance evaluations of the testbed experiments and the simulations are discussed in Section 4.5.

5.2 System Model

In this section, we describe the system model which is used for testbeds and simulations. A WASN can be modeled with a UDG $G(V, E)$ where V is the set of nodes and E is the set of edges. There exists an edge between any two nodes u and v if and only if the Euclidean distance between u and v is less than or equal to T_r. N_i denotes the neighbors of node i, $S_i \in \{IN, OUT\}$ denotes the state of node i, c_i denotes the capacity of node i, and id_i denotes the identifier of node i. The system can start from any configuration. Thus, the initial values of the local variables are generated randomly. We have made the following assumptions in this work:

1. Each node has a distinct *id*. Thus, the network is *id*-based.

2. The capacity of each node is uniform, and this type is more suitable for homogeneous networks. The proposed algorithm can run even if the capacity is a non-uniform.

3. Communication links between nodes are bidirectional.

4. All nodes are homogeneously equipped except the sink node.

5. Each node knows its neighbors within its T_r, and T_r is same for each node. However, the proposed algorithm can run even if T_r is different.

6. Each node executes the same program. Thus, the proposed algorithm is uniform.

7. The rules of the proposed algorithm are executed atomically.

8. An unfair distributed scheduler is used as a runtime scheduler.

9. As the communication model, a message passing model is used.

10. If the topology changes due to nodes joining or leaving the network, a new CapDS should be constructed.

5.3 Proposed Algorithm

The steps of the proposed algorithm called A_{CapDS} are given in Algorithm 5.1. The local state S_i of node i can have two variable states: OUT and IN. Node i with $S_i = \text{OUT}$ means that it is out of the CapDS. If it is a member of CapDS, S_i sets as IN. $Dominator_i$ is the dominator node of node i. The null value is shown by $\perp$. $Dominatees_i$ is the set of dominatees of node i. If the capacity of node i is full, and i is the dominator of all dominatees in $Dominatees_i$ then $IsFull_i$ equals to true. Otherwise, $IsFull_i$ equals to false. $EmptyCapacity_i$ is used to calculate the maximum number of additional dominatee assignment to the dominator node i in order to fulfill its capacity. $CanDominatees_i$ is the set of candidate dominatees that can be assigned to dominator node i. $CanDominators_i$ is the set of possible dominators one of which can be chosen as the dominator node for dominatee node

i. $MinNbr_i$ and $MaxNbr_i$ show the *id* of minimum and maximum neighbor *id*s of node i, respectively. $IsFullMac_i$ is used to set the value of $IsFull_i$ variable.

The algorithm has 8 rules (Rs) where R1, R2, R3, R4, and R8 are executed by IN nodes, and the other rules are executed by OUT nodes. R1, R7, and R8 are designed to construct a DS, and the other rules are given to provide the capacity constraint. According to R1, if node i has IN state, its capacity is not full, and it has an IN neighbor having lower *id* and $isFull = false$ then node i changes to state OUT and sets its $Dominator_i$ variable to null. Any two IN nodes of whose capacities are not full cannot be neighbors, and this property is supported by R1. If the state of node i is IN, and the size of the dominatees set of i is greater than the capacity then it executes R2 and removes the nodes from its dominatees set until this set is not greater than the capacity. In the initial state, the size of $Dominatees_i$ can overflow its capacity. R2 solves this problem.

By R3, dominators choose their dominatees considering their capacities. If the size of the dominatees set of an IN node i is lower than its capacity, and there exits at least one candidate dominatee which has not chosen a dominator yet, it executes R3 and adds the candidate dominatees into its dominatees set until its capacity is full. According to R4, if the dominator node i includes node j in its dominatees set where node j points to another dominator node then node i excludes j from its dominatees set.

If the state of node i is OUT, the dominator variable of i is null, and there exists at least one IN neighbor which includes i in its dominatees set, then node i executes R5 by choosing a dominator from its CanDominators set and setting its dominator variable. A dominatee which has not a dominator and chosen by a dominator, answers this dominator request and completes a matching by R5. According to R6, if the state of node i is OUT, the dominator variable of node i is not null, and i is not in the dominatees set of its dominator or the state of its dominator is OUT or its dominator is not a neighbor of it, then node i sets the dominator variable to null. If there is a wrong matching between a dominator and a dominatee, R4 and R6 solve this problem.

According to R7, if the state of node i is OUT, it does not point to a dominator, each dominatee neighbor of i with lower *id* points to a dominator, and each dominator neighbor of node i has full capacity then node i becomes a dominator by changing its state to IN. R7 supports that there must be a dominator in every closed neighborhood except fulfilled dominators and its dominatees. $IsFull_i$ provides communication between a dominator and its neighbors to inform whether the capacity of the dominator is full. If none of the rules mentioned so far are true, and $IsFull_i$ variable is incorrect then node i updates it by executing R8.

An example operation of A_{CapDS} is shown in Figure 5.1. Black nodes denote IN state, and white nodes denote OUT state where the capacity is uniform and equal to 2. Each dominatee points its dominator with an arrow. The initial state of the system is shown in Figure 5.1.a. Initially, nodes 1 and 5 are IN, and the other nodes are OUT. Nodes 2, 3, and 4 have not a dominator except node 6. $IsFull_5$ is true but it is incorrect. The stabilized system is shown in Figure 5.1.b. Nodes 1 and 4 are in CapDS, and their $isFull$ variables are true. The dominator of nodes 2 and 3 is node 1, and the dominator of nodes 5 and 6 is node 4. We can see the detailed convergence steps of A_{CapDS} in Table 5.1.

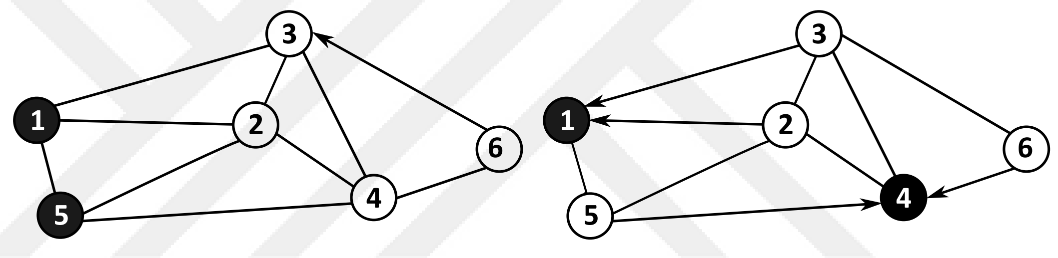

Figure 5.1 An example execution of A_{CapDS} algorithm.

Table 5.1 Convergence steps of A_{CapDS} in Figure 5.1

	Initial States	**Step 1**	**Step 2**	**Step 3**	**Step 4**
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 $Dominator_6 = 4$	

Algorithm 5.1 A$_{\text{CapDS}}$

Inputs.

id_i: *The identifier of node i.*

N_i: *The neighbors of node i.*

c_i: *The capacity of node i.*

Variables.

S_i: *The state of node i.*

$Dominator_i$: *The dominator of node i.*

$Dominatees_i$: *The dominatees set of node i.*

$IsFull_i$: *The variable of node i that shows whether its capacity is full or not.*

Macros.

$EmptyCapacity_i: \left|c_i - |Dominatees_i|\right|$.

$CanDominatees_i: \{j \in N_i | S_j = OUT \wedge Dominator_j = \perp \wedge j \notin Dominatees_i\}$.

$CanDominators_i: \{j \in N_i | S_j = IN \wedge i \in Dominatees_j \wedge EmptyCapacity_j \geq 0\}$.

$MinNbr_i: \min\{j \in N_i\}$, $MaxNbr_i: \max\{j \in N_i\}$.

$MaxEmptyNbr_i: j \in N_i | S_j = IN \wedge \forall t \in N_i (S_t = IN \wedge j \neq t \wedge EmptyCapacity_j \geq EmptyCapacity_t)$.

$IsFullMac_i \in \{true, false\}$:

- $\boldsymbol{if}$ $(S_i = IN \wedge EmptyCapacity_i = 0 \wedge \forall j \in Dominatees_i [Dominator_j = i]$ $\boldsymbol{then}$ $IsFullMac_i = true$.
- $\boldsymbol{if}$ $S_i = OUT \vee \left(S_i = IN \wedge \left(EmptyCapacity_i \neq 0 \vee \exists j \in Dominatees_i [S_j = IN \vee Dominator_j \neq i]\right)\right)$ $\boldsymbol{then}$ $IsFullMac_i = false$.

Rules.

R1. 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**
  $S_i = OUT, Dominator_i = \perp$

R2. 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 $S_i \neq IN \vee |Dominatees_i| \leq c_i$
  $IsFull_i := IsFullMac_i$

R3. if $S_i = IN \wedge EmptyCapacity_i > 0 \wedge CanDominatees_i \neq \emptyset$ **then**
  repeat
    $Pick\ MinNbr_i \in CanDominatees_i$
    $Dominatees_i := Dominatees_i \cup \{MinNbr_i\}$
    $CanDominatees_i := CanDominatees_i \backslash \{MinNbr_i\}$
  until $S_i \neq IN \vee EmptyCapacity_i = 0 \vee CanDominatees_i = \emptyset$
  $IsFull_i := IsFullMac_i$

R4. 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 $S_i \neq IN \vee \forall j \in Dominatees_i [(Dominator_j = i \vee Dominator_j = \perp) \wedge j \in N_i \wedge S_j \neq IN]$
  $IsFull_i := IsFullMac_i$

R5. if $S_i = OUT \wedge Dominator_i = \perp \wedge CanDominators_i \neq \emptyset$ **then**
  $Pick\ MaxEmptyNbr_i\ from\ CanDominators_i$
  $Dominator_i := MaxEmptyNbr_i$

R6. 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**
  $Dominator_i := \perp$

R7. if $S_i = OUT \wedge Dominator_i = \perp \wedge \forall j \in N_i \left[\left(S_j = OUT \wedge \left(i < j \vee Dominator_j \neq \perp\right)\right) \vee (S_j = IN \wedge IsFull_j)\right]$ **then**
  $S_i := IN,\ Dominatees_i := \emptyset, IsFull_i := IsFullMac_i$

R8. if $S_i = IN \wedge \neg(R1 - R7) \wedge IsFull_i \neq IsFullMac_i$ **then**
  $IsFull_i := IsFullMac_i$

5.4 Theoretical Analysis

5.4.1 Closure

Lemma 5.1 *When the system is stable,* $Dominator_i = j$ *if and only if* $i \in Dominatees_j$.

Proof. Assume, by contradiction, that the system is stable, and $Dominator_i = j$ but $i \notin Dominatees_j$. In this case, node i executes R6. If $i \in Dominatees_j$ and $Dominator_i \neq j$, it causes two cases. In case 1, if $Dominator_i = \perp$, node i executes R5. In case 2, if $Dominator_i \neq \perp$ and $Dominator_i \neq j$, node j executes R4. Since there is at least one move in a stable system, it is a contradiction.

Theorem 5.1 *In any state in which no node is enabled the set* S *is a CapDS.*

Proof. Suppose to the contrary that the system is stable, and no node is enabled but S is not a *CapDS*. Then either (*i*) S is not a DS or (*ii*) S is a DS but not capacitated. In case (*i*), if node i has the least *id* in its neighborhood or all of its OUT neighbors with a lower *id* has a dominator and $IsFull$ variables of all its IN neighbors are true, R7 is enabled. If there exists at least an OUT neighbor node with a lower *id* which has not a dominator, R5 or R7 is enabled for it. This contradicts to the assumptions that no node is enabled. Now consider (ii), let node j is a node in S, and the capacity of j is overflow. In this situation, R2 is enabled. No rule lets the number of elements of $Dominatees_j$ be more than the capacity. Any node not in S must have at least one IN neighbor since S is a DS. If node i does not have a dominator, and the capacity of node j (a neighbor of node i) is not full, then node j executes R3. If the capacity of node j is full, i or one of its OUT neighbors with a lower *id* which does not have a dominator neighbor executes R7. We contradict our assumption. Lemma 5.1 provides that (dominator, dominatee) pairs are matched correctly then no node is enabled. Our theorem holds true.

5.4.2 Convergence

Lemma 5.2 $IsFull$ of an IN node is exactly correct after the first move.

Proof. In the first move, all IN nodes know the dominators of all OUT neighbors. Let j is an IN node. If the size of $Dominatees_j$ is equal to the capacity, the dominator of all OUT neighbors in $Dominatees_j$ is j and $IsFull_j = false$, it changes $IsFull_j$ to true. Otherwise, it is false. No rule lets the $IsFull_j$ to be incorrect after the first move.

Lemma 5.3 *A node executes R8 at most twice, and the first one must be the first move and the last one must be the last move.*

Proof. Let j is an IN node in graph G. In the initial configuration, if $IsFull_j$ is correct and true, then there is no rule to force j to execute any rule. In order that a dominatee node i in $Dominatees_j$ can execute R6, j must execute R3 to remove i from $Dominatees_j$. On the other hand, i must execute R6 so that j can execute R3. There is a deadlock between i and j. In this case, if $IsFull_j$ is correct and true in the first move it stays so. Now suppose that $IsFull_j$ is initially incorrect, and j does not execute any rule from R1 to R7 then node j can execute R8, and $IsFull_j$ will be correct after the first move by Lemma 5.2. If $IsFull_j$ is true after the first move it stays so. If j executes R8 in any step after the first move, it can execute no rule. Because, when the capacities of nodes in $Dominatees_j$ are full, and all dominatees in $Dominatees_j$ choose j as dominator, it changes $IsFull_j$ to true from false. There is no rule lets $IsFull_j$ to be false again. Thus, a node executes R8 at most twice, and the first one must be the first move and the last one must be the last move.

Lemma 5.4 *A node can execute R2 at most once and as the first move.*

Proof. In the initial configuration, the number of $Dominatees_j$ of IN node j can overflow the capacity. In this case, R2 can be executed once as the first move, and the number of elements of $Dominatees_j$ can be at most equal to the capacity. No rule lets the number of elements of $Dominatees_j$ be more than the capacity. So, R2 can be executed at most once and as the first move.

Lemma 5.5 *A node executes R7 at most twice and R1 at most once. Only the following two sequences of states and their suffixes are possible during the execution of A_{CapDS} under an unfair distributed scheduler.*

OUT IN OUT

OUT IN OUT IN

Proof. In the initial configuration, suppose that $Dominator_i$ of an OUT node i is null, $IsFull$ variables of all IN neighbors with lower *id* are true and incorrect, the dominators of all OUT neighbors with lower *id*s are not null. In this case, i executes R7 and changes state to IN, and any IN neighbor with lower *id* executes R8 and changes its $IsFull$ variable to false. Node i executes R1 as the second move and changes state to OUT again. Node i makes sequence OUT IN OUT. In order to execute R7 again, all IN neighbors of i with lower *id* must execute R8 and make their $IsFull$ variables to true, and all OUT neighbors with lower *id* must choose their dominator with R5. In this case, i executes R7 second time and remains so. Because IN neighbors do not execute any rule after the second R8 by Lemma 5.2. On the other hand, even if any OUT neighbor with lower *id* assigns dominator variable to null with R6, it cannot force i to run R1. At the end of these processes, i makes sequence OUT IN OUT IN.

Suppose now that node i has initially state IN if there exists a neighbor node j with lower *id*, state IN and $IsFull_j = false$, i executes R1 and changes state to OUT. If i does not execute R7, and any IN neighbor j adds i into $Dominatees_j$, i executes R5 and does not execute any rule. At the end of these processes, i makes sequence IN OUT.

Lemma 5.6 *A node executes R5 and R6 at most* n^2 *times until the system is stable.*

Proof. Suppose that node i has initially state OUT, it can execute R5, R6 or R7. If $Dominator_i$ is not null, and i is not in the dominatees set of $Dominator_i$, it executes R6 as the first move and sets $Dominator_i$ as null. After the first move, if i has an IN neighbor (j) which includes i in $Dominatees_j$, i executes R5. Node j can execute R1 and change state OUT. In this case, i executes R6 again. These moves (R6-R5) can repeat as long as i has IN neighbors which includes i in their $Dominatees$ set and make (IN-OUT) move with R1. There must be an IN neighbor (k) which has the minimum id among its IN neighbors and remains as IN. If k adds i into $Dominatees_k$, and i chooses k as dominator, i cannot execute any rule because there is no rule which breaks the matched of i and k. If k does not add i, an OUT neighbor (u) that executes R1 before can execute R7 and add i into $Dominatees_u$ by R3, and u cannot execute R1 again by Lemma 5.5. Then node i chooses u as a dominator and cannot execute any rule because there is no rule which breaks the matched of i and u. Node i can make OUT IN OUT sequence and stays so by Lemma 5.5. In this situation, we must multiply the total move count for R5 and R6 by 2. The following formula shows the greatest move count of R5 and R6 where n is the set of nodes of graph G:

$$= 4xy(x = \{S_i = OUT\}, y = \{S_j = IN\}, n = x + y)$$

$$= 4x(n - x)$$

$$= 4nx - 4x^2$$

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

Lemma 5.7 *R3 and R4 can be executed at most* $\frac{2n^2}{3}$ *times until the system is stable.*

Proof. In the initial configuration, the system may not be stable. Suppose that the state of node i is initially IN and there are n nodes. In the initial configuration, let X is the set of nodes in CapDS, Y is the set of other nodes, $|X| = x$ and $|Y| = y$. Any node in Y has a neighbor of at least one node in CapDS. In the first step, x nodes can execute R3, and all of them can add the same node into their $Dominatees$. In the second step, at least one node of Y executes R5. So, the capacity of every node in X must be one and equal in the worst-case scenario. In the third step $(x - 1)$ nodes can execute R4 and remove the matched node in Y from their $Dominatees$. In the fourth step, $(x - 1)$ nodes can execute R3 and add the same node into their $Dominatees$ from Y except the matched node. In the fifth step, at least one node of Y executes R5 and chooses a dominator from X. Node i can make IN OUT IN sequence and stays so by Lemma 5.5. In this situation, we must multiply the total move count for R3 and R4 by 2. It is shown below with equations that how many times the dominators execute R3 and R4 totally until the system is stable.

$$x + y = n(x \geq 1, y \geq 1, n \geq 2$$

$$x + (x - 1) + (x - 1) + (x - 2) + (x - 2) + \cdots$$

$$= \begin{cases} x + 2\sum_{i=1}^{x-1}(x - i), & x \leq y \ (5.1) \\ x + 2\sum_{i=1}^{y-1}(x - i) + x - y, & x > y \ (5.2) \end{cases}$$

$$Case\ 1: if\ x \leq y$$

$$x + 2\sum_{i=1}^{x-1}(x - i)$$

$$= x + 2\left(\sum_{i=1}^{x-1} x - \sum_{i=1}^{x-1} i\right)$$

$$= x^2$$

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

$$Case\ 2: if\ x > y$$

$$x + 2\sum_{i=1}^{y-1}(x - i) + x - y$$

$$= x + 2\left(\sum_{i=1}^{y-1} x - \sum_{i=1}^{y-1} i\right) + x - y$$

$$= 2x - y + 2x(y - 1) - y(y - 1)$$

$$= 2y(n - y) - y^2$$

$$f(y) = 2ny - 3y^2$$

$$f(y)' = 2n - 6y = 0$$

$$y_{max} = \frac{n}{3}\ and\ \frac{2n^2}{3}\ is\ the\ greatest\ move\ count.$$

$Since\ \frac{n^2}{2} < \frac{2n^2}{3}\ for\ n \geq 0,\quad so\ until\ being\ stable\ R3\ and\ R4\ can\ be$ $executed\ at\ most\ \frac{2n^2}{3}\ times.$

Theorem 5.2 *A_{CapDS} is self-stabilizing under an unfair distributed scheduler and stabilizes after at most* $\left(\frac{5n^2}{3} + 6n\right)$ *moves with a CapDS.*

Proof. Any node can execute R1 at most once and R7 twice by Lemma 5.5. It causes at most $3n$ moves. From Lemma 5.4, any node executes R2 at most once. Thus, n moves can be executed totally at most by R2. R3 and R4 can be executed at most $\frac{2n^2}{3}$ times by the nodes from Lemma 5.7. R5 and R6 can be executed at most n^2 times by the nodes from Lemma 5.6. R8 can be executed at most $2n$ times by nodes from Lemma 5.3. So that the total move count is bounded by $\left(\frac{5n^2}{3} + 6n\right)$

Theorem 5.3 *A_{CapDS} returns a solution of the minimum CapDS problem with approximation ratio 6 for a UDG model of nodes having a uniform capacity.*

Proof. Suppose that the set D^* refers to the minimum CapDS of V of UDG G. We can formulate the size of D^* as $|D^*| = V/(c+1)$. Let S be the CapDS produced by A_{CapDS}. Suppose that S' is the set of nodes in S whose degrees are more than or equal to the capacity, S'' is the set of other nodes in S. The nodes in S' can dominate at most c nodes. The nodes in S'' creates a maximal independent set. Since no IN nodes in S'' can be a neighbor of each other, and every OUT node has an IN neighbor in S''. Obviously, $IsFull$ variable of an IN node in S'' cannot be true due to its degree is lower than the capacity. Any node in MIS can dominate at most 5 nodes in a UDG (Alzoubi et al., 2002). Since $S' \leq D^*$ and $S'' \leq 5D^*,\ S \leq 6D^*$, A_{CapDS} is a 6-approximation algorithm.

5.5 Performance Evaluation

5.5.1 Testbed experiments

We evaluated A_{CapDS} by testbed experiments including IRIS sensor nodes. We generated topologies varying from 10 to 40 nodes by augmenting 10 nodes at each step in our laboratory environment. The topologies are simple, connected and undirected, and each node has a unique *id*. Three classes of densities are used as sparse, medium, and dense where average degrees of these topologies are 4, 6, 8, respectively. IRIS motes have 2.4 GHz IEEE 802.15.4 compliant transceiver, 250 kbps data rate, 8 kB RAM, 128 kB programmable flash memory. The proposed algorithm is written in NesC language supported by TinyOS. We calculated the move count, received byte count, and energy consumption. Each measurement is produced by averaging 10 repeated testbed experiments. All nodes initially send a Hello message to their 1-hop neighbors to inform them about their initial states. If node i is dominator, it sends its id_i, S_i, $IsFull_i$, and

$Dominatees_i$ variables in a message while a dominatee sends its id_i, S_i, and $Dominator_i$ variables. Each node in the network can run the algorithm simultaneously. If the preconditions of a rule are satisfied, the node can move to execute its rule. When a node moves, it broadcasts the new state to its neighbors. When there is no enabled node, the network is stabilized, and a CapDS is created. This means that each dominatee has a dominator, and each dominator has a set of dominatees of whose size does not overflow the capacity.

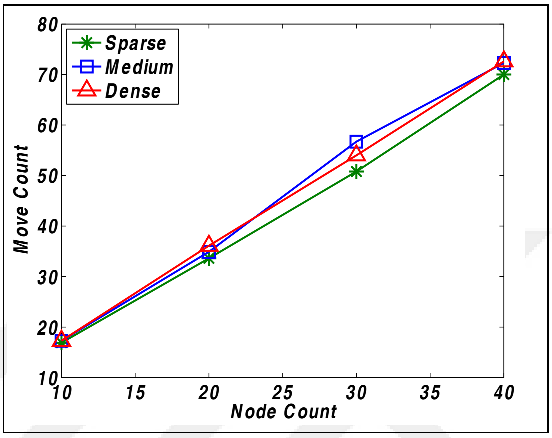

Figure 5.2 Move count of A_{CapDS} against node count and density.

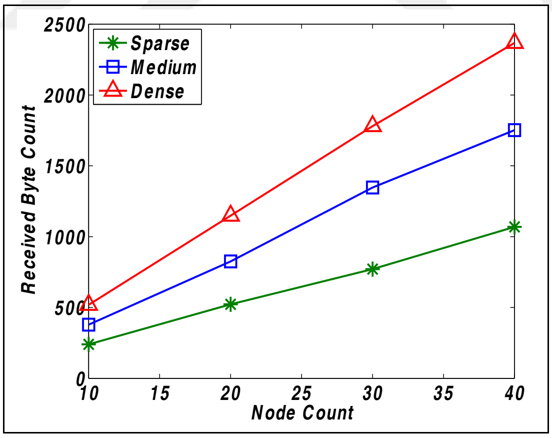

Figure 5.3 Received byte count of A_{CapDS} against node count and density.

Move count of A_{CapDS} against node count and density is shown in Figure 5.2. Move count directly affects the received byte count and energy consumption since each node sends its new state information to its neighbors more frequently. When node count is increased, move count values are increasing linearly. Besides, move count values are generally stable against varying density values. Figure 5.3 represents the received byte count of A_{CapDS} against node count and density. Average degree is greater in dense topologies since the number of connections between nodes is higher. According to the measurements, the received byte count increases linearly when node count and density increases.

In Figure 5.4, the energy consumption measurements of A_{CapDS} against node count and density are given. The amount of energy consumption is calculated for each node from Formula 4.5. The results of energy consumption are similar to the received byte count which is one of the most significant criteria affecting energy consumption. Therefore, the energy consumption increases proportionally with the node count and density. These measurements taken from the testbed of IRIS nodes show us that our algorithm consumes resources reasonably, and it is stable against varying node count and degree values.

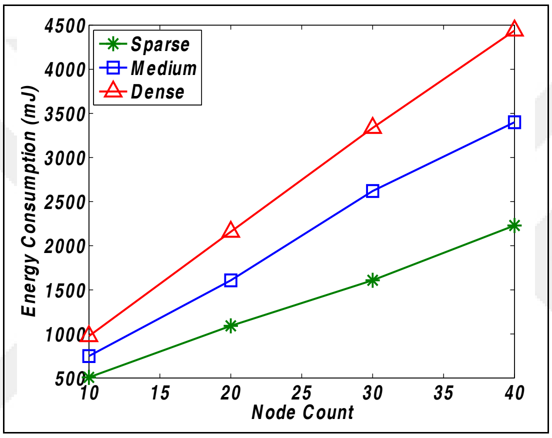

Figure 5.4 Energy consumption of A_{CapDS} against node count and density.

5.5.2 Simulations

Since we are limited with 40 sensor nodes for testbed experiments, we make simulations to measure the behavior of the algorithm in large-scale networks and to compare it with the other approaches. As aforementioned, to the best of our knowledge, there is no distributed self-stabilizing capacitated DS algorithm in the literature. Therefore, we designed two self-stabilizing CapDS algorithms from Chiu et al.'s $4n$ move self-stabilizing DS algorithm (Chiu et al., 2014) by applying a hierarchical collateral composition technique (Datta, 2013). In the first algorithm called as C_{Random}, we used a random approach in which the dominatees choose randomly its dominator, and the dominators definitively add them into their dominatees set. On the other hand, the second algorithm called as C_{Greedy} uses a minimum *id* priority-based approach in which the dominatees always choose the dominator with minimum *id* and the dominators add definitively them into their dominatees set. We tested A_{CapDS}, C_{Random}, and C_{Greedy} on TOSSIM which is a discrete event simulator for TinyOS sensor network. We generated undirected and random graphs. The topologies are changing from 50 to 250 nodes by augmenting 50 nodes at each step. The densities of topologies are classified into three categories as sparse, medium, and dense where the average degrees of the nodes are approximately 4, 6, and 8, respectively. Each measurement is produced by averaging 50 repeated simulations. We compared algorithms against node count and the average degree in terms of coefficient of variation (CV), move count, received byte count and energy consumption. The CV is formulated in Formula 5.3 which is a measure of relative variability. It is used to represent the balance of the clusters that are formed by cluster heads (dominators) and cluster members (dominatees).

$$CV = \left(\frac{Standard\ Deviation}{Mean}\right) x100 \qquad (5.3)$$

A comparison of the CV of algorithms against node count is shown in Figure 5.5.a. As the node count increases, CV decreases since dominatee nodes have more dominator neighbors for an assignment which provides to construct more balanced clusters. The CV of A_{CapDS} is the smallest among the algorithms. A_{CapDS} is averagely 1.49 times better than C_{Random} and 1.55 times better than C_{Greedy}. Figure 5.5.b shows the CV values of algorithms against average degree. When the average degree increases, the size of CapDS reduces that leads to a decrease in CV values. A_{CapDS} has far better performance than its counterparts which have similar results like Figure 5.5.a. As a result, CV of A_{CapDS} is significantly better than its competitors against node count and average degree.

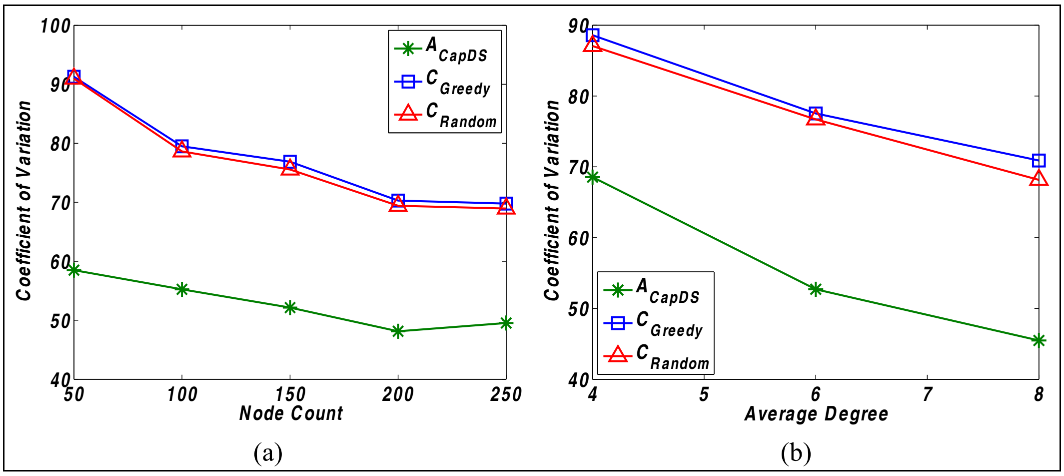

Figure 5.5 CV of algorithms against a) Node count b) Average Degree.

Move count is one of the most significant criteria affecting energy consumption and decreasing move count is vital to prolonging the network lifetime. Figure 5.6.a represents the move count of algorithms against node count. Move counts of the algorithms increase proportionally with the node count where A_{CapDS} is 1.61 times better than C_{Random} and 1.60 times better than C_{Greedy}. In Figure 5.6.b, the move count of algorithms against the average degree is given. A_{CapDS} has undoubtedly the best performance than the other algorithms on various topologies. These results show us that the move count of A_{CapDS} is significantly smaller than its counterparts.

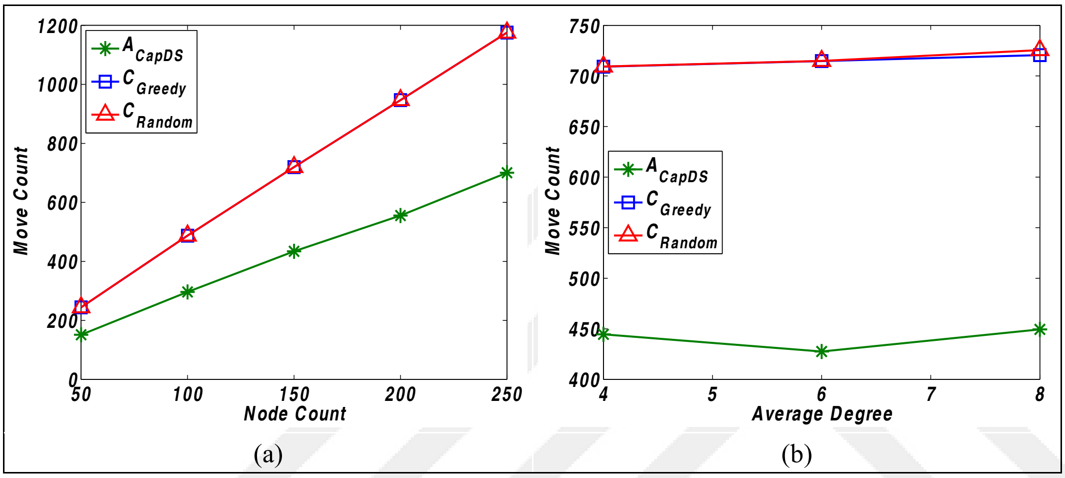

Figure 5.6 Move count of algorithms against a) Node count b) Average Degree.

Figure 5.7.a shows the received byte count of algorithms against node count. We can see that the received byte count values of the algorithms generally increase linearly while node count increases. A_{CapDS} is 1.13 times better than C_{Random} and 1.09 times better C_{Greedy}. The results of both other algorithms are very similar. The received byte count values of the algorithms against node degree are given in Figure 5.7.b. Again, the values have a linear increase since the average degree of the nodes is higher in dense graphs and a transmitted message of node i is received by all neighbors of node i. A_{CapDS} has obviously the best performance among the algorithms in terms of received byte count. Therefore, it uses efficiently sources by reducing transmitted and received byte count.

Energy efficiency has paramount importance for WASNs to prolong the network lifetime. We calculated the energy consumption from Formula 4.5. Figure 5.8.a shows the energy consumption of algorithms against node count. Energy consumption values of all algorithms increase as expected while node count increases. A_{CapDS} has the best energy performance where A_{CapDS} is 1.14

times better than C_{Random} and 1.10 times better than C_{Greedy}. In Figure 5.8.b, the energy consumption values of the algorithms against the average degree are represented. A_{CapDS} has the best energy efficiency. The performance of C_{Greedy} is slightly better than C_{Random}. Consequently, the measurements taken from the simulations reveal us that the proposed algorithm outperforms its competitors according to CV, move count, received byte count and energy consumption against various node counts and densities. Moreover, A_{CapDS} significantly prolongs the lifetime of the network by providing energy efficiency.

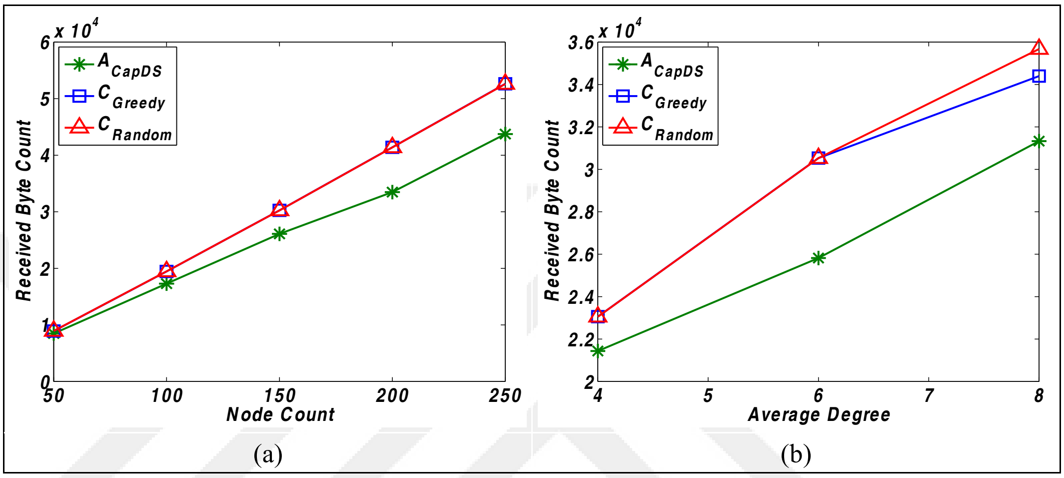

Figure 5.7 Received byte count of algorithms against a) Node count b) Average Degree.

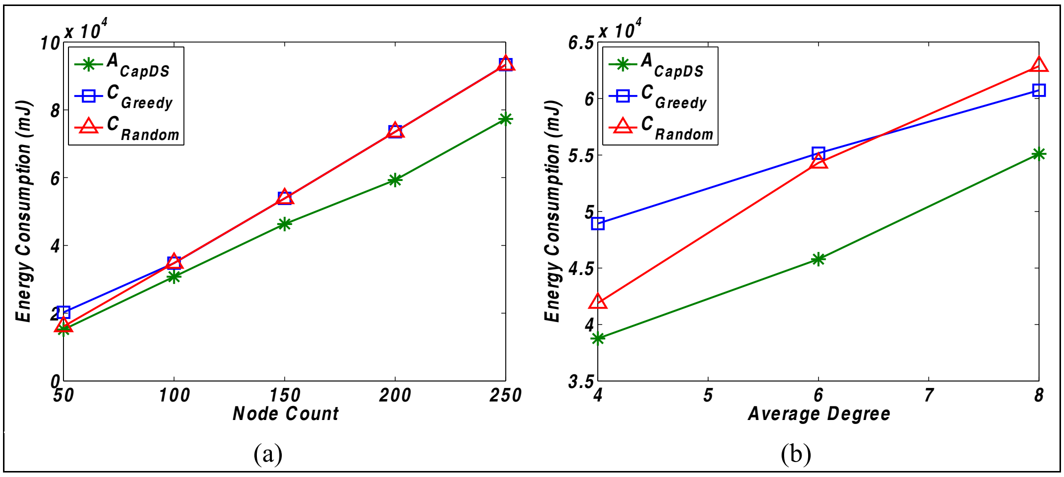

Figure 5.8 Energy consumption of algorithms against a) Node count b) Average Degree.

A_{CapDS} can cope with uniform and non-uniform capacity but we can give a 6-approximation ratio for only uniform capacity. It is considered important that the nodes with less battery life are not loaded much in clustering. The network lifetime in WASN using clustering can be defined as the time beginning of the experiment until the first node in the dominating set failure. We calculate the network lifetime of algorithms against node count and average degree. The number of data bytes sent in one packet may not exceed 127 bytes in the packet structure of IEEE 802.15.4. E_{max} denotes the maximum consumed energy for a packet of which size is 127 bytes, and E_i denotes the energy of node i. Initially, we set random energy to the nodes of which energies are varied between $1000E_{max}$ and $10000E_{max}$. We supposed that when a CapDS constructed by algorithms, each dominatees of a dominator sends a packet of which size is 127 bytes in every round. The lifetime of a dominator node i denoted L_i is calculated from Formula 4.6 as $\min\{L_i\}$. D_i denotes the degree of node i, and c_i denotes the capacity of node i. E_{avg} and D_{avg} denote the average energy and degree of the neighbors, respectively. We calculate the initial capacity of the nodes considering their energy from Formula 4.3 and Formula 4.4.

Figure 5.9.a presents the network lifetime of algorithms against node count. If the node count increases, the time to first node failure generally decreases. Because when the node count increases, the probability of being a dominator node with less energy increases. A_{CapDS} has the best network lifetime since it is 1.86 times better than C_{Random} and 1.69 times better than C_{Greedy}. In Figure 5.9.b, the network lifetime of algorithms against the average degree is shown. The density of a network affects the degrees of nodes and the size of CapDS. Thus, when the average degree increases, the time to first node failure decreases. A_{CapDS} has the best among the algorithms. The results show that even if the capacity is non-uniform, A_{CapDS} prolongs the network lifetime the best for realistic experiments modeled by homogenous or heterogenous networks.

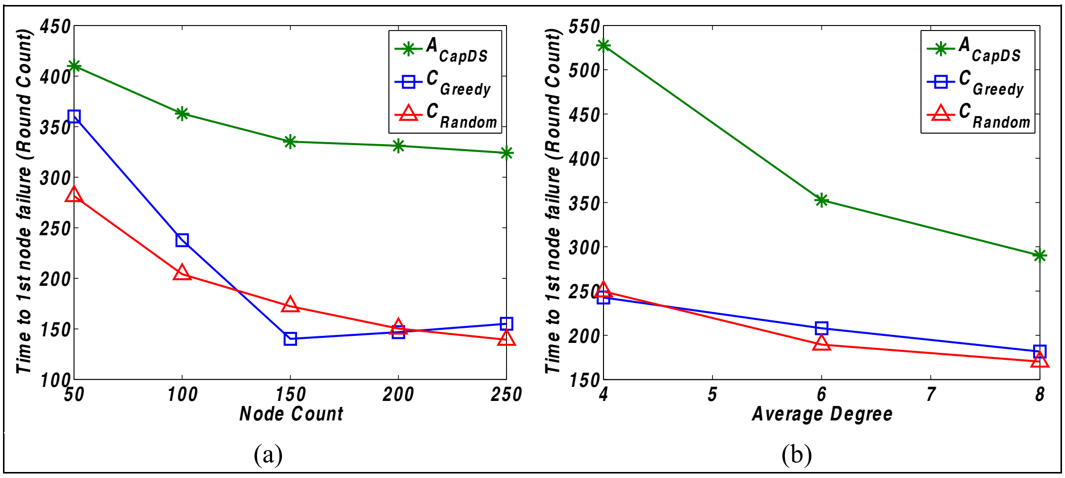

Figure 5.9 Network lifetime of algorithms against a) Node count b) Average Degree.

6. A DISTRIBUTED SELF-STABILIZING ALGORITHM FOR CAPACITATED CONNECTED DOMINATING SET PROBLEM

6.1 Introduction

Energy efficiency is one of the major issues in WASNs which lack a fixed infrastructure and centralized control. In order to prolong the network lifetime, connected dominating set (CDS) has been widely used as a virtual backbone in WASNs. The sensor nodes in WASNs can be failed due to lack of battery, have hardware damage, link failure or environmental interference. Therefore, it is desirable to design an energy efficient and fault-tolerant CDS algorithm in WASNs. A non-masking fault tolerance method denoted self-stabilizing tolerates any finite number of transient faults.

In this thesis, we propose the first distributed self-stabilizing algorithm for CapCDS construction in WASNs. It stabilizes at most $\left(\frac{n^2}{3} + 2n\right)$ moves under an unfair distributed scheduler where n is the number of nodes. We supposed that a CDS is constructed by a self-stabilizing distributed CDS algorithm like (Kamei et al., 2016) before. The remaining of this paper is organized as follows. Section 6.2 represents the system model. The proposed algorithm is shown in Section 6.3. Section 6.4 is devoted to the theoretical analysis of it. The performance evaluation that includes the results of real experiments and simulations is finally given in Section 6.5.

6.2 System Model

The topology of a distributed system can be represented as a simple, connected and undirected graph $G = (V, E)$ where V and E represent the set of vertices and the set of edges, respectively. The identifier of a node i is denoted id_i, and we assume that each node has a unique id. Any two nodes i and j are neighbor if there is an edge between i and j. The transmission range of the nodes is the same in UDGs. There is an edge between i and j if their transmission range covers the center of each other in UDGs. N_i denotes the set of neighbor nodes of node i. Each node executes the same program. Thus, the proposed algorithm is uniform. In order to understand clearly the proposed algorithm, we have made the following assumptions:

1. Each node has a distinct id and the capacity of each node is non-uniform. Communication links between nodes are bidirectional.

2. All nodes are homogeneously equipped except the sink node. Each node knows its neighbors within its T_r.

3. The proposed algorithm is uniform, and the rules of the proposed algorithm are executed atomically.

4. An unfair distributed scheduler is used as a runtime scheduler.

5. Message passing model is used as a communication model.

6. The nodes can join or leave the network, the new CapCDS should be constructed since the algorithm is self-stabilizing.

An example CapCDS on a UDG of which nodes have the same transmission range dotted by circles is shown in Figure 6.1. The dominators (nodes 2, 4, and 8) in CapCDS are colored black and the dominatees (nodes 1, 3, 5, 6, 7, and 9) out of CapCDS are colored white. The edge of the arrows represents the dominator of a dominatee. The capacity of each dominator node i is denoted c_i, the dominator of a dominatee j is denoted $Dominator_j$, and the set of dominatees of a dominator node i is denoted $Dominatees_i$. The size of $Dominatees_i$ is represented $|Dominatees_i|$. In this work, we assume the capacity is non-uniform and greater than or equal to 1, and it represents the maximum size of $|Dominatees_i|$ of the dominator node i.

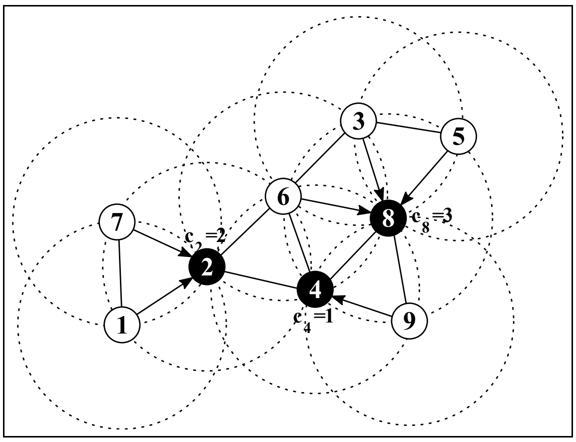

Figure 6.1 An example CapCDS on a sample UDG.

6.3 Proposed Algorithm

The proposed algorithm called A_{CapCDS} is distributed and self-stabilizing. A_{CapCDS} shown in Algorithm 6.1 is formed by rule sets and executed atomically in steps. The rules are assigned a number in priority order. We can separate the rules of the algorithm as dominator rules and dominatee rules. The first three rules are for dominators, and the last three rules are for dominatees. If a node is in CDS, the state of it is IN, otherwise OUT1 or OUT2 which are simply OUT. S_i denotes the state of node i. If a dominatee node has a dominator in CDS, its state is OUT1. If the capacity of all dominator neighbor nodes of a dominatee is full, the state of this node is OUT2. An OUT2 node cannot dominate any OUT node. When the system is stabilized, the set of A_{CapCDS} exists union of IN and OUT2 nodes. $NumInNbr_i$ denotes the number of IN neighbor nodes of an OUT node i, and $EmptyCapacity_i$ denotes the empty space of $Dominatees_i$. These macros support a balanced capacity matching between dominators and dominatees. $EmptyCapacity_i$ is equal to (-1) if the size of $Dominatees_i$ overflow the c_i, and $|EmptyCapacity_i|$ denotes the size of $EmptyCapacity_i$. The null value is shown $\perp$. If no node is enabled in any state, the system is stabilized, each dominatee has a dominator, and the capacity of any dominators is not overflow.

The algorithm works as follows. A self-stabilizing system can start from any configuration in the initial state. Thus, the size of the dominatees set of any dominator node j can initially overflow its capacity. In this situation, Rule 1 (R1) is enabled. When R1 is executed, the dominatee nodes in $Dominatees_j$ are removed until the size of $Dominatees_j$ is not overflow the capacity according to $MaxNumInNbr_i$. If there is equality for $MinNumInNbr_i$, $MaxNumInNbr_i$, $MaxEmptyNbr_i$ macros, the symmetry is broken by minimum *id* priority. If the size of dominatees set of an IN node j is not full, and j has at least one OUT1 or OUT2 node i which has not just chosen its dominator, it executes Rule 2 (R2) and adds i into $Dominatees_j$ from $CanDominatees_j$ according to $MinNumInNbr_i$ until the capacity of j is full. If an IN node j has at least a dominatee node i in $Dominatees_j$ of which dominator is not j and not null or not in its neighborhood or $S_j = IN$, it executes Rule 3 (R3) and removes all i nodes from $Dominatees_j$.

If an OUT1 or OUT2 node i has not a dominator, and there exists an IN dominator node j which has added i into $Dominatees_j$, it executes Rule 4 (R4) and chooses j node as a dominator according to $MaxEmptyNbr_i$. If the state of node i is OUT2, it changes its state to OUT1. If the dominator of an OUT1 node i

is null, and there is no node j which adds i into $Dominatees_j$, it executes Rule 5 (R5) and changes its state to OUT2. If there is an OUT1 or OUT2 dominatee of which dominator is not null, and it is not in the dominatees set of its dominator or its dominator is not in its neighborhood or the state of its dominator is not IN, it executes Rule 6 (R6) and changes its dominator as null. When the system is stabilized, there are IN, OUT1, and OUT2 nodes in the system, and CapCDS exists. IN and OUT2 nodes are in CapCDS but only IN nodes in CDS.

An example execution of A_{CapCDS} on a sample UDG is presented in Figure 6.2. In Figure 6.2.a, the initial configuration is shown. Each node has a unique *id* and a non-uniform capacity. We assume that CDS is constructed with nodes 1 and 3 before. The state of black nodes is IN, the state of white nodes is OUT1, and the state of grey nodes are OUT2. Our algorithm runs in steps under an unfair distributed scheduler. In the first step, node 1 executes R2 and adds node 8 into $Dominatees_1$. Nodes 2, 5, and 7 execute R6 and set their dominators as ⊥. Node 3 executes R1 and excludes 4 from $Dominatees_3$ to make the capacity not overflow. Node 6 executes R4 and set its $Dominator$ to node 3. Node 8 executes R5 and set its state to OUT2. In the second step, node 1 executes R3 and excludes node 3 from $Dominatees_1$. Nodes 2, 7, and 8 execute R4, node 2 sets its $Dominator_2$ to node 3, and the others set their $Dominator$ variables to node 1. Node 5 executes R5 and sets its state to OUT2. In the third step, node 1 executes R2 and adds node 5 into $Dominatees_1$. In the fourth step, node 5 executes R4 and sets $Dominator_5$ as node 1. Then the system is stabilized. The convergence steps of A_{CapCDS} in Figure 6.2 is illustrated in Table 6.1. The stabilized system configuration is shown in Figure 6.2.b, and CapCDS is constructed from IN nodes (1 and 3) and OUT2 node (4).

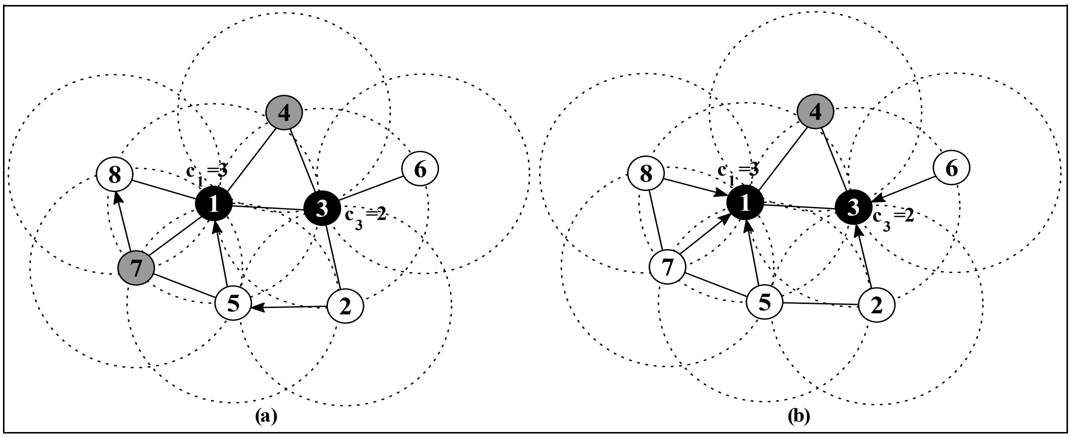

Figure 6.2 An example execution of A_{CapCDS} algorithm

Algorithm 6.1 A_{CapCDS}

Inputs.

id_i: *The identifier of node i.*

N_i: *The neighbors of node i.*

c_i: *The capacity of node i.*

Variables.

$S_i \in \{IN, OUT1 \text{ or } OUT2\}$: *The state of node i.*

If node i is in both CDS and CapCDS, $S_i = IN$.

If node i is only in CapCDS, $S_i = OUT2$.

If node i is out of both CDS and CapCDS, $S_i = OUT1$.

$Dominator_i$: *The dominator of node i.*

$Dominatees_i$: *The dominatees set of node i.*

Macros.

$EmptyCapacity_i: \left|c_i - |Dominatees_i|\right|$.

$CanDominatees_i: \{j \in N_i | S_j \neq IN \ \wedge Dominator_j = \perp \wedge j \notin Dominatees_i\}$.

$CanDominators_i: \{j \in N_i | S_j = IN \ \wedge i \in Dominatees_j \wedge EmptyCapacity_j \geq 0\}$.

$NumInNbr_i: |\{j \in N_i | S_j = IN\}|$.

$MinNumInNbr_i: j \in N_i | S_j \neq IN \ \wedge \ \forall t \in N_i(S_t \neq IN \ \wedge \ j \neq t \wedge NumInNbr_t \geq NumInNbr_j)$.

$MaxNumInNbr_i: j \in N_i | S_j \neq IN \ \wedge \ \forall t \in N_i(S_t \neq IN \ \wedge \ j \neq t \wedge NumInNbr_t \leq NumInNbr_j)$.

$MaxEmptyNbr_i: j \in N_i | S_j = IN \ \wedge \ \forall t \in N_i(S_t = IN \ \wedge \ j \neq t \wedge EmptyCapacity_j \geq EmptyCapacity_t)$.

Rules.

R1. if $S_i = IN \wedge |Dominatees_i| > c_i$ **then**

repeat

$Pick\ MaxNumInNbr_i \in Dominatees_i$

$Dominatees_i \coloneqq Dominatees_i \backslash \{MaxNumInNbr_i\}$

until $S_i \neq IN \vee |Dominatees_i| \leq c_i$

R2. if $S_i = IN \wedge \ EmptyCapacity_i > 0 \wedge CanDominatees_i \neq \emptyset$ **then**

repeat

$Pick\ MinNumInNbr_i \in CanDominatees_i$

$Dominatees_i \coloneqq Dominatees_i \cup \{MinNumInNbr_i\}$

$CanDominatees_i \coloneqq CanDominatees_i \backslash \{MinNumInNbr_i\}$

until $S_i \neq IN \vee \ EmptyCapacity_i = 0 \vee CanDominatees_i = \emptyset$

R3. 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

$Dominatees_i \coloneqq Dominatees_i \backslash \{j\}$

until $S_i \neq IN \vee \ \forall j \in Dominatees_i[(Dominator_j = i \vee Dominator_j = \perp) \wedge j \in N_i \wedge S_j \neq IN]$

R4. if $S_i \neq IN \wedge \ Dominator_i = \perp \wedge CanDominators_i \neq \emptyset$ **then**

$Pick\ MaxEmptyNbr_i\ from\ CanDominators_i$

$Dominator_i \coloneqq MaxEmptyNbr_i$

if $S_i = OUT2$ **then** $S_i = OUT1$

R5. if $S_i = OUT1 \wedge \ Dominator_i = \perp \wedge CanDominators_i = \emptyset$ **then**

$S_i = OUT2$

R6. If $S_i \neq IN \wedge Dominator_i \neq \perp \wedge \left[i \notin Dominatees_{Dominator_i} \vee Dominator_i \notin N_i \vee S_{Dominator_i} \neq IN\right]$ **then**

$Dominator_i = \perp$

Table 6.1 Convergence steps of A_{CapCDS} in Figure 6.2.

	Initial States	Step 1	Step 2	Step 3	Step 4
Node 1	$S_1 = IN$ $Dominatees_1 = \{3,7\}$	R2 $Dominatees_1 = \{3,7,8\}$	R3 $Dominatees_1 = \{7,8\}$	R2 $Dominatees_1 = \{5,7,8\}$	
Node 2	$S_2 = OUT1$ $Dominator_2 = 5$ $NumInNbr_2 = 1$	R6 $Dominator_2 = \perp$	R4 $Dominator_2 = 3$		
Node 3	$S_3 = IN$ $Dominatees_3 = \{2,4,6\}$	R1 $Dominatees_3 = \{2,6\}$			
Node 4	$S_4 = OUT2$ $Dominator_4 = \perp$ $NumInNbr_4 = 2$				
Node 5	$S_5 = OUT1$ $Dominator_5 = 1$ $NumInNbr_5 = 1$	R6 $Dominator_5 = \perp$	R5 $S_5 = OUT2$		R4 $Dominator_5 = 1$
Node 6	$S_6 = OUT1$ $Dominator_6 = \perp$ $NumInNbr_6 = 1$	R4 $Dominator_6 = 3$			
Node 7	$S_7 = OUT2$ $Dominator_7 = 8$ $NumInNbr_7 = 1$	R6 $Dominator_7 = \perp$	R4 $S_7 = OUT1$ $Dominator_7 = 1$		
Node 8	$S_8 = OUT1$ $Dominator_8 = \perp$ $NumInNbr_8 = 1$	R5 $S_8 = OUT2$	R4 $S_8 = OUT1$ $Dominator_8 = 1$		

6.4 Theoretical Analysis

In this section, we show the proof of correctness of A_{CapCDS}. The general requirement to prove the correctness of a self-stabilizing algorithm is to show that it has closure and convergence properties.

6.4.1 Closure

Lemma 6.1 *When the system is stable,* $Dominator_i = j$ *if and only if* $i \in Dominatees_j$.

Proof. Assume, by contradiction, that the system is stable, and $Dominator_i = j$ but $i \notin Dominatees_j$. In this case, node i executes R6. If $i \in Dominatees_j$ and $Dominator_i \neq j$, it causes two cases. In case 1, if $Dominator_i = \perp$, node i executes R4. In case 2, if $Dominator_i \neq \perp$ and $Dominator_i \neq j$, node j executes R3. Since there is at least one move in a stable system, it is a contradiction.

Lemma 6.2 *When the system is stable, any node with state* $OUT2$ *remains so.*

Proof. Suppose that the system is stable. Any node with state $\mathrm{OUT2}$ can execute only R4 or R6 if dominator neighbors execute R2 or R3. However, any neighbor dominator nodes cannot execute R2 or R3 because the system is stable.

6.4.2 Convergence

Lemma 6.3 *Any node can execute R1 at most once and as the first move.*

Proof. In the initial configuration, the number of $Dominatees_j$ set can overflow the capacity. In this case, R1 can be executed once in the first step, and the number of elements of $Dominatees_j$ can be at most the capacity. No rule let the number of elements of $Dominatees_j$ be more than the capacity. So, R1 can be executed at most once and as the first move.

Lemma 6.4 *Any node executes R4 at most once.*

Proof. After i executes R4, i cannot execute R4 or R5 until it makes the dominator as null because it has a dominator. It must execute R6 to make the dominator as null. Besides, node j must remove node i from $Dominatees_j$ to be executed R6. Node j must execute R1 or R3 to remove node i. Node j cannot execute R1 since node i cannot execute R4 while the capacity of node j is overflow. On the other hand, node i must make the dominator as null to be executed R3. In this situation, there is a deadlock between nodes i and j because R3 is the precondition for R6 and R6 is the precondition for R3. Therefore, a node executes R4 at most once.

Lemma 6.5 *Any node can execute R5 at most once.*

Proof. Suppose that node i executes R5 in any step. R4 must be executed to be executed R5 again. If R4 is executed, node i does not make a move by Lemma 6.4. Thus, any node can execute R5 at most once.

Lemma 6.6 *Any node can execute R6 at most once.*

Proof. A self-stabilizing system can be started from any initial configuration. In any step $Dominator_i = j$ and $i \notin Dominatees_j$ can be true. In this situation, node i executes R6 and makes its dominator as null. It must be matched by R4 again to execute R6 again. No rule can be executed after R4 by Lemma 6.4.

If node j matched with node i executes R1 as the first move and removes i from $Dominatees_j$, i executes R6 and make the dominator as null. Node i must execute R4 and choose its dominator to be executed R6 again. A dominator node k must execute R2 and add i into $Dominatees_k$ to execute R4 again. No rule can be executed after R4 by Lemma 6.4.

Lemma 6.7 *R2 and R3 can be executed at most* $\frac{n^2}{3}$ *times until the system is stable.*

Proof. Suppose that there are n nodes in a UDG $G(V,E)$. In the initial configuration, X is the set of nodes in CDS, and x is the size of X. Y is the set of nodes out of CDS, and y is the size of Y. Any node in Y has a neighbor of one node in CDS at least. In the first step, x nodes can execute R2, and all of them can add the same node into their *Dominatees*. In the second step, at least one node of Y executes R4. So, at least one dominator and one dominatee match and remain so by Lemma 6.4. Thus, the capacity of every node in X must be one and equal in the worst-case scenario. In the third step $(x-1)$ nodes can execute R3 and remove the matched node in Y from their $Dominatees$. In the fourth step, $(x-1)$ nodes can execute R2 and add the same node into their $Dominatees$ from Y except the matched node. In the fifth step, at least one node of Y executes R4 and chooses a dominator from X. It is shown below with formulas that how many times the dominators execute R2 and R3 totally until the system is stable.

$$x + y = n(x \geq 1, y \geq 1, n \geq 2)$$

$$x + (x-1) + (x-1) + (x-2) + (x-2) + \cdots$$

$$= \begin{cases} x + 2\sum_{i=1}^{x-1}(x-i), & x \leq y \ (6.1) \\ x + 2\sum_{i=1}^{y-1}(x-i) + x - y, & x > y \ (6.2) \end{cases}$$

$$Case\ 1: if\ x \leq y$$

$$x + 2\sum_{i=1}^{x-1}(x-i)$$

$$= x + 2\left(\sum_{i=1}^{x-1} x - \sum_{i=1}^{x-1} i\right)$$

$$= x^2$$

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

$$Case\ 2: if\ x > y$$

$$x + 2\sum_{i=1}^{y-1} (x - i) + x - y$$

$$= x + 2\left(\sum_{i=1}^{y-1} x - \sum_{i=1}^{y-1} i\right) + x - y$$

$$= 2y(n - y) - y^2$$

$$f(y) = 2ny - 3y^2$$

$$f(y)' = 2n - 6y = 0$$

$$y_{max} = \frac{n}{3} \text{ and } \frac{n^2}{3} \text{ is the greatest move count.}$$

$$Since\ \frac{n^2}{4} < \frac{n^2}{3}\ for\ n \geq 0, \qquad so\ until\ being\ stable\ R2\ and\ R3\ can\ be$$

$$executed\ at\ most\ \frac{n^2}{3}\ times.$$

Theorem 6.1 *A_{CapCDS} is self-stabilizing under an unfair distributed scheduler and stabilizes after at most $\left(\frac{n^2}{3} + 2n\right)$ moves with a capacitated connected dominating set, where n is the number of nodes.*

Proof. In order to calculate the time complexity of A_{CapCDS} in the worst-case scenario, the following initial assumptions are required:

- CDS has been established by (Kamei et al., 2016) before. However, CapCDS is not.

- There are n nodes in a simple, connected and undirected UDG $G(V,E)$ with a unique *id*.

- The states of all nodes out of CDS are OUT1, and all nodes out of CDS are connected to all nodes in CDS.

- The number of nodes in CDS is $(2n/3)$, the number of nodes out of CDS is $(n/3)$, and the capacity of every dominator node is uniform and equal to one by Lemma 6.7.

- The size of $Dominatees_j$ of every node in CDS is the overflow of their capacity.

- The dominator of each OUT1 node is not null and their dominators are not in N_i.

When these assumptions are true in the initial configuration, in the first step all dominators execute R1, and they correct their capacity of $Dominatees$ with $(2n/3)$ moves; all dominatees execute R6 and make their dominator as null with $(n/3)$ moves then they cannot execute R6 again by Lemma 6.6. In the second step, (2n/3) dominators execute R2 and can add the same dominatee not chosen by a dominator into their $Dominatees_j$; (n/3) dominatees execute R5 and change their state OUT2. In the third step, the node added into $Dominatees_j$ by all dominators executes R4 and chooses one of them as a dominator, and it cannot make a move by Lemma 6.4. In the fourth step $((2n/3)-1)$ dominators execute R3, in the fifth step $((2n/3)-1)$ dominators execute R2, in the sixth step one of the dominatees execute R4, and these steps loop until all dominatees choose a dominator. In the last step, $(2n/3)$ dominators execute R3, and the system stabilizes. The time complexity formula of the worst-case scenario is shown below:

$$\frac{2n}{3}+\frac{n}{3}+\frac{2n}{3}+\frac{n}{3}+1+\left(\frac{2n}{3}-1\right)+\left(\frac{2n}{3}-1\right)+1+\cdots+$$

$$\left(\frac{2n}{3}-\left(\frac{n}{3}-1\right)\right)+\left(\frac{2n}{3}-\left(\frac{n}{3}-1\right)\right)+1+\frac{2n}{3}$$

$$=\frac{8n}{3}+\sum_{i=1}^{\frac{n}{3}}1+2\sum_{i=1}^{\frac{n}{3}-1}\frac{2n}{3}-i$$

$$= \frac{8n}{3} + \frac{n}{3} + 2\left(\sum_{i=1}^{\frac{n}{3}-1} \frac{2n}{3} - \sum_{i=1}^{\frac{n}{3}-1} i\right)$$

$$= 3n + 2\left(\left(\frac{n}{3} - 1\right)\left(\frac{2n}{3}\right) - \frac{\left(\frac{n}{3} - 1\right)\left(\frac{n}{3}\right)}{2}\right)$$

$$= 3n + \left(\frac{n^2}{3} - n\right)$$

$$= \frac{n^2}{3} + 2n$$

6.5 Performance Evaluation

6.5.1 Testbed experiments

The testbed experiments presented in this subsection use from 10 to 40 IRIS motes based on the ATmega1281 microcontroller and increased by 10 in a laboratory environment. IRIS motes have 2.4 GHz IEEE 802.15.4 compliant transceiver, 250 kbps data rate, 8 kB RAM, 128 kB programmable flash memory. A_{CapCDS} is written in NesC language supported by TinyOS and tested on TOSSIM with simple, connected and undirected UDGs which are generated randomly. The topologies are classified in three densities which are sparse, medium, and dense where average degrees of these topologies are 4, 6, and 8, respectively. Java-based gateway software is developed to listen to the motes in the testbeds via a sink node connected to a notebook.

Move count, transmitted byte count, received byte count and energy consumption are measured, and each measurement is produced by averaging 10 repeated testbed experiments. Firstly, a CDS is constructed by (Kamei et al., 2016). $Dominatees_i$ and $Dominator_j$ variables are randomly initiated for the dominators i and the dominatees j, respectively. In order to reduce the packet interference probability, a carrier sense multiple access with collision avoidance MAC protocol is used. The dominators send id_i, S_i, c_i, and $Dominatees_i$ variables where the dominatees send id_j, S_j, $NumInNbr_j$, and $Dominator_j$ variables in a message packet if they are chosen by an unfair distributed scheduler and move after sending Hello message.

We used non-uniform capacity for the testbeds and simulations. In the structure of IEEE 802.15.4, the maximum data bytes sending in a message packet do not exceed 127 bytes. E_{max} represents the maximum consumed energy for sending a packet of which size is 127 bytes, and E_i represents the energy of node i. In the beginning, we set random energy to each node of which energy is varying between $1000xE_{max}$ and $10000xE_{max}$. D_i denotes the degree of node i, and c_i denotes the capacity of node i. E_{avg} and D_{avg} denote the average energy and degree of the neighbors, respectively. We calculate the initial capacity of the nodes considering their energy from Formula 4.3 and Formula 4.4.

Move count is an important criterion affecting the transmitted byte count and received byte count directly. Because when a node moves, it sends its new local variables in a message packet, and the message is received by its neighbors. Decreasing move count provides energy efficiency and extends the network lifetime. As illustrated in Figure 6.3.a, if the node count rises, move count rises linearly. On the other hand, the rising of the density affects move count in a direct proportion.

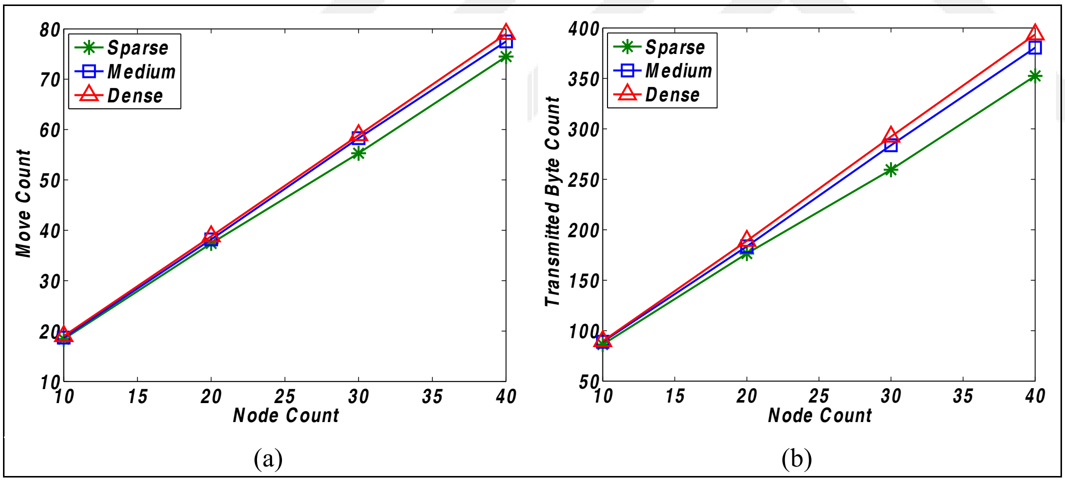

Figure 6.3 a) Move count b) Transmitted byte count of A_{CapCDS} against node count and density.

Transmitted byte count of A_{CapCDS} against node count and density is shown in Figure 6.3.b. The results are similar to Figure 6.3.a since each move causes the transmitted byte count increases. If the node count increases, the transmitted byte count increases. The transmitted byte count is at most in dense graphs and at least in sparse graphs. The reason behind this behavior, the sparse graph has less move to match the dominators and the dominatees due to CDS size is greater in sparse graphs than dense graphs.

In WASNs, most of the energy is consumed for communication by the antennas. Thus, the transmitted byte count and received byte count affects directly the energy consumption. As shown in Figure 6.4.a, the rising of the node count rises received byte count linearly. Since the message sent by a moved node is received by the neighbor nodes, and the size of neighbors is greater in the dense graphs than the sparse graphs, the received byte count rises when the density rises.

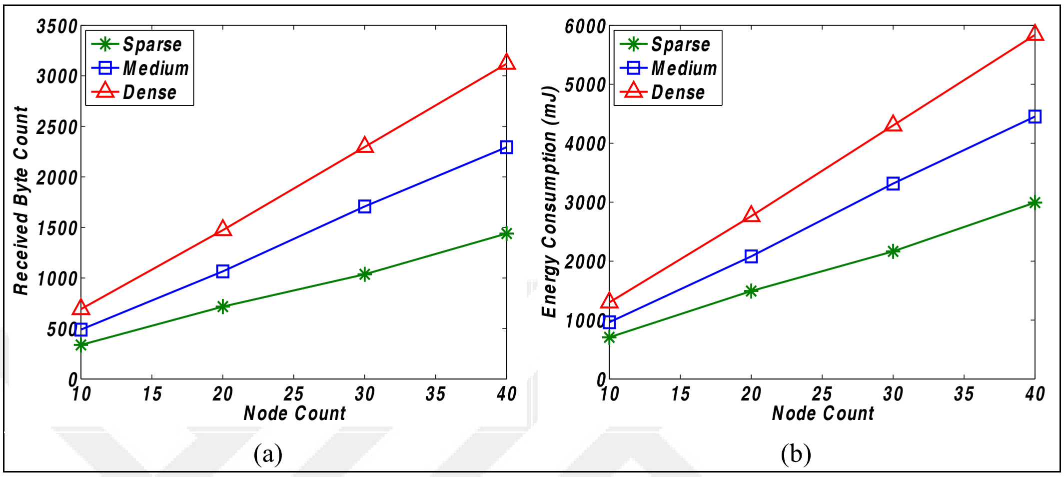

Figure 6.4 a) Received byte count b) Energy consumption of A_{CapCDS} against node count and density.

Energy efficiency is one of the most important criteria in WASNs since the nodes have bounded energy. We calculated the energy consumption from Formula 4.5. Energy consumption (mJ) of A_{CapCDS} against node count and density is shown in Figure 6.4.b. It is clearly shown that when the node count or density rise, energy consumption generally presents a linear rise. Consequently, the testbed experiments with IRIS nodes on various topologies show that A_{CapCDS} reacts well to the stability and scalability against various node counts and densities.

6.5.2 Simulations

In this subsection, we evaluate the performance of A_{CapCDS} on a discrete event simulator TOSSIM under an unfair distributed scheduler. The simulation analysis of WASNs can be presented with UDGs. The UDG topologies are simple, connected, undirected, and randomly generated in a field of $(1000x1000)m^2$, and each node has a unique *id*. The number of nodes is changing from 50 to 250 by increasing 50 at each step, and three type network densities are used as sparse, medium, and dense of which average degrees are approximately 4, 6, and 8, respectively. Nodes are assigned with the initial non-uniform capacity that is randomly generated with Formula 4.3 or 4.4. Each measurement is the average of 50 repeated simulations. Initially, a CDS is constructed by (Kamei et al., 2016).

The initial variables of $Dominatees$ and $Dominator$ are randomly generated. In the beginning, the dominators broadcast id_i, S_i, c_i, and $Dominatees_i$ variables where the dominatees broadcast id_j, S_j, $NumInNbr_j$, and $Dominator_j$ variables in a Hello message packet. Then they send these packets if they are selected by an unfair distributed scheduler to move until the system is stabilized. In order to evaluate the performance of A_{CapCDS}, we measure move count, transmitted byte count, received byte count, energy consumption, and the lifetime of the networks.

Move count is one of the most significant criteria affecting energy consumption since decreasing move count prolongs the network lifetime. When a node moves, it broadcasts the new variables to its neighbors within one-hop distance. Thus, a move causes that the transmitted byte count and the received byte count increase. As indicated in Figure 6.5.a, the move count of the nodes using different densities increases with the increase in the node count. Although the results are similar according to the various densities, move count is at least for sparse graphs. As demonstrated in Figure 6.5.b, the transmitted byte count is compatible with the move count shown in Figure 6.5.a. As soon as the node count or the density increases, the transmitted byte count tends to increase linearly.

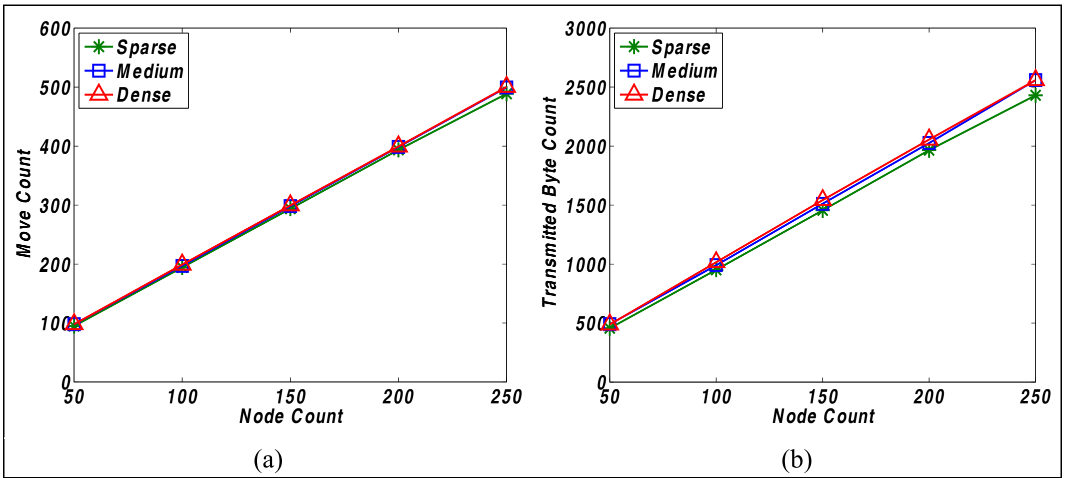

Figure 6.5 a) Move count b) Transmitted byte count of A_{CapCDS} against node count and density.

Figure 6.6.a shows the received byte count of A_{CapCDS} against node count for various densities. The average degree of the network affects directly the received

byte count. When the density or node count increase, the received byte count increases directly. Energy consumption of A_{CapCDS} against node count for various densities is presented in Figure 6.6.b. We calculated the energy consumption from Formula 4.5 since transmitting or receiving the messages is a fundamental consumer in a message passing communication model in WASNs. When the node count is enlarged, the energy consumption increases in a natural way. The energy consumption of A_{CapCDS} increases linearly when the density increases. It is at most in the dense topologies since the average degree is the highest in them.

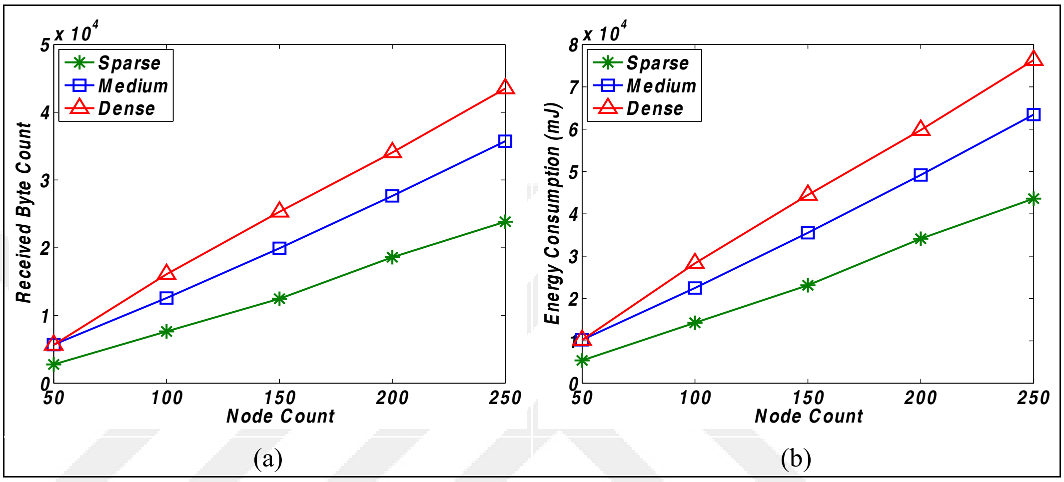

Figure 6.6 a) Received byte count b) Energy consumption of A_{CapCDS} against node count and density.

The network lifetime is a critical issue in WASNs. The lifetime of a sensor network is commonly defined as the time from the network starts execution to the first node failure. To the best of our knowledge, there is no distributed self-stabilizing CapCDS algorithm in the literature. Thus, we designed two distributed self-stabilizing CapCDS algorithms to compare them with A_{CapCDS} in terms of the network lifetime. We assume that a CDS is constructed by (Kamei et al., 2016) before. The first algorithm is called Greedy, and the main aspect of it is that the dominatees nodes choose a dominator node in the CDS which has minimum *id* from the set of IN neighbor nodes. The second algorithm is called Random, and the dominatees choose randomly their dominators from the set of IN neighbor nodes. In both algorithms, the dominators add definitively the dominatees which choose them into their $Dominatees$. If there is a wrong match, the dominators remove the dominatees from their $Dominatees$, and the dominatees set their dominator variable as null. The lifetime of a dominator node i denoted L_i is calculated from Formula 4.6 as $\min(L_i)$.

The comparison of the algorithms in terms of the network lifetime against node count is shown in 6.7.a. When the node count increases, the network lifetime decreases generally since the probability of being a dominator node with less energy increases. A_{CapCDS} has the best network lifetime where A_{CapCDS} is 1.72 times better than Random and 2.12 times better than Greedy.

In Figure 6.7.b, the network lifetime of the algorithms against the average degree is presented. The CapCDS size is directly affected by the network density. It is smaller in dense graphs than sparse graphs. So, the lifetime of the algorithms decreases when the density increases. A_{CapCDS} has the best performance in terms of the lifetime and significantly outperforms its counterparts.

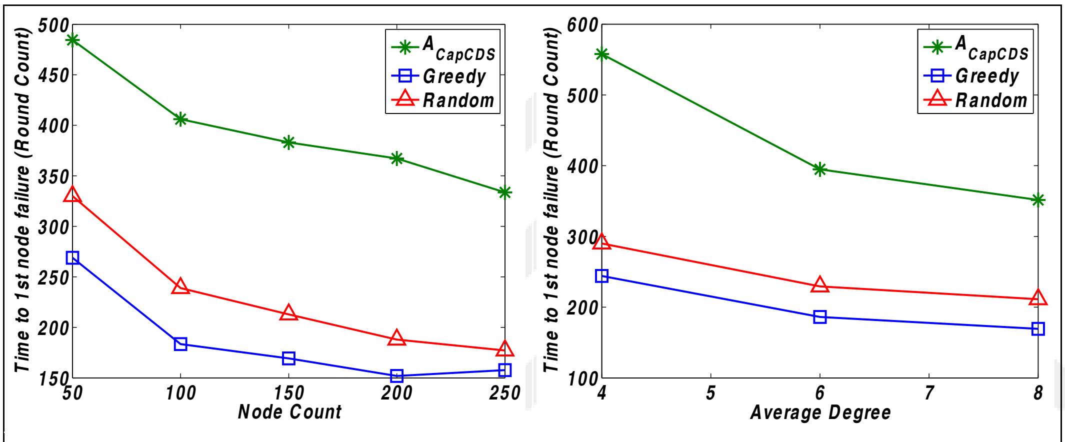

Figure 6.7 a) Lifetime of A_{CapCDS} b) Lifetime of algorithms against node count and density.

Consequently, the simulation results are evidence that A_{CapCDS} is stable and scalable despite large-scale networks and various network densities. The simulation results are compatible with the testbed results. Moreover, A_{CapCDS} copes with non-uniform capacities under an unfair distributed scheduler. Thus, it is more suitable for real applications. Although the system starts randomly in any state for testbeds and simulations, a CapCDS is always constructed by the proposed algorithm. Since there is no distributed self-stabilizing CapCDS algorithm in the literature, to the best of our knowledge, we proposed two approaches which are Greedy and Random. A_{CapCDS} is more energy efficient than its counterparts since the distribution of dominatees according to the capacity of dominators is more balanced in A_{CapCDS}. Therefore, it provides load-balancing along energy efficiency. Since a CapCDS is a virtual backbone in WASNs, A_{CapCDS} prolongs outstandingly the network lifetime.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis, we proposed three new distributed self-stabilizing algorithms for the capacitated domination problems which are maximal independent set, dominating set, and connected dominating set. These problems have a cornerstone role for many important applications such as clustering, routing, data aggregation, topology control, and building other graph structures. To the best of our knowledge, proposed algorithms are the first distributed self-stabilizing algorithms for these problems in the literature.

All proposed algorithms are uniform and work on *id*-based networks. They can cope with non-uniform capacity, but we can give a 6-approximation ratio for A_{CapDS} if it works with uniform capacity on UDGs. The uniform capacity is more suitable for homogeneous networks where the non-uniform capacity is more suitable for heterogeneous networks. In this work, the capacity is a metric of a dominator node i which shows an upper bound for the size of the dominatees set which is dominated by dominator node i. A dominatee node can be dominated at most one dominator. However, vice versa depends on the capacity of a dominator. These features of the capacity provide load-balancing in WASNs by matching dominators and dominatees.

The proposed algorithms are tested on IRIS motes via testbeds which contain from 10 to 40 motes increasing by 10 at each step. UDG topologies are used, and they are simple, connected and undirected. The densities of the networks are classified into three types which are sparse, medium, and dense of which average degrees are approximately 4, 6, and 8, respectively. In order to test the performance of them on large-scale networks, they are also tested on TOSSIM via simulations which contain from 50 to 250 nodes increasing by 50 at each step. The proposed algorithms are as follows:

We proposed the first distributed self-stabilizing algorithm called A_{CapMIS} for a soft capacitated maximal independent set problem. The algorithm stabilizes an unstable system at most $\left(\frac{5n^2}{6} + 3n\right)$ moves under an unfair distributed scheduler. A_{CapMIS} is theoretically proved in terms of convergence and closure. Then the performance of A_{CapMIS} through testbeds and simulations is evaluated practically on the UDG topologies. Although UDGs are used as the network model for

A_{CapMIS}, the proposed algorithm is applicable for CapMIS construction when nodes in a WASN have different transmission ranges. We compare A_{CapMIS} with two approaches which are generated a hierarchical collateral composition technique and called A_{Random} and A_{Greedy}. The results show us that A_{CapMIS} reacts well to the stability and scalability against various node counts and densities. Moreover, A_{CapMIS} is more energy efficient than its counterparts and prolongs significantly the network lifetime.

The second proposed algorithm is the first distributed self-stabilizing 6-approximation algorithm called A_{CapDS} for hard capacitated dominating set problem. A_{CapDS} stabilizes at most $\left(\frac{5n^2}{3} + 6n\right)$ moves under an unfair distributed scheduler. A_{CapDS} is theoretically proved in terms of convergence and closure. We also show through testbed experiments and simulations that our algorithm is favorable. A_{CapDS} is compared with two approaches which are generated a hierarchical collateral composition technique and called C_{Random} and C_{Greedy}. It has the best performance and energy efficiency. Thus, it prolongs more the lifetime of the network than its counterparts. Determination of the expected move count, reducing the total move count in the worst-case, redefining capacity as a function of network parameters such as throughput, energy, reliability, and designing an approximation algorithm running on WASNs modeled as undirected graphs of nodes with non-uniform capacities are open problems.

The last proposed algorithm called A_{CapCDS} is the first distributed self-stabilizing hard CapCDS algorithm. It is considered significant that the nodes with less battery life are not loaded much in WASNs. Thus, the proposed algorithm forms a capacitated virtual backbone via a CapCDS. A_{CapCDS} gains a legitimate configuration at most $\left(\frac{n^2}{3} + 2n\right)$ moves using an unfair distributed scheduler. Additionally, it is easy that A_{CapCDS} can be composed with any distributed self-stabilizing algorithms via a hierarchical collateral composition technique. The results are obviously demonstrated that A_{CapCDS} reacts well to the stability and scalability against various node counts and densities. Moreover, it is compared with two CapCDS approaches which are Greedy and Random proposed in this thesis, and A_{CapCDS} is more energy efficient than its counterparts and significantly prolongs the network lifetime in WASNs.

Finally, all proposed distributed self-stabilizing algorithms for capacitated domination problems react well to the stability, scalability, and load-balancing in WASNs. We hope that they will be followed up by many researchers in the future.

7.2 Future Work

Applying a capacity constraint to other graph problems such as vertex cover, matching, spanning tree or other types of domination problems via a distributed self-stabilizing algorithm is an attractive research direction. We intend to design randomized algorithms for capacitated domination problems for anonymous networks.

Designing capacitated distributed self-stabilizing algorithms by using composition techniques for capacitated domination problems is open. Some difficult graph problems can be solved in an efficient way by using them. For instance, a CapCDS can be constructed by the composition of two self-stabilizing algorithms for CapMIS and BFS problems. Moreover, a new composition technique can be designed for capacitated domination problems.

The popularity of IoT and social networks which are distributed systems are increasing day by day. We focused on WASNs in this thesis. A deeper and more comprehensive experimental analysis could be possible for IoTs and social networks in order to apply practically the proposed algorithms in realistic environments.

REFERENCES

Akyildiz, I.F., Melodia, T., and Chowdhury, K.R., 2007, A survey on wireless multimedia sensor networks, *Computer Networks*, 51:91-690pp.

Alon, N., Babai, L., and Itai, A., 1986, A fast and simple randomized parallel algorithm for the maximal independent set problem, *Journal of Algorithms*, 7(4):567-583pp.

Alzoubi, K.M., Wan, P.-J., and Frieder, O., 2002, Distributed heuristics for connected dominating sets in wireless ad hoc networks, *Journal of Communications and Networks*, 4(1):22-29pp.

Alzoubi, K.M., Wan, P.-J., and Frieder, O., 2003, Maximal independent set, weakly-connected dominating set, and induced spanners in wireless ad hoc networks, *International Journal of Foundations of Computer Science*, 14(2):287-303pp.

Arapoglu, O., Akram, V.K., and Dagdeviren, O., 2019, An energy-efficient, self-stabilizing and distributed algorithm for maximal independent set construction in wireless ad hoc and sensor networks, *Computer Standards and Interfaces*, 62:32-42pp.

Barilan, J., Kortsarz, G., and Peleg, D., 1993, How to allocate network centers, *Journal of Algorithms*, 15(3):385-415pp.

Basagni, S., 2001, Finding a maximal weighted independent set in wireless networks, *Telecommunication Systems*, 18(1):155-168pp.

Becker, A., 2016, Capacitated Dominating Set on Planar Graphs, CoRR, abs/1604.04664.

Blelloch, G.E., Fineman, J.T., and Shun, J., 2012, Greedy sequential maximal independent set and matching are parallel on average, in Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA ’12, (New York, NY, USA), 308-317pp.

Bodlaender, H.L., Lokshtanov, D., and Penninkx, E., 2009, Parameterized and exact computation., Heidelberg: Springer-Verlag, Planar Capacitated Dominating Set Is W[1]-Hard, 50-60pp.

Cheng, X., Ding, M., Du, H., and Jia, X., 2006, Virtual backbone construction in multihop ad hoc wireless networks, *Wireless Communications and Mobile Computing*, 6:183-190pp.

Chiu, W.Y., Chen, C., and Tsai, S.Y., 2014, A 4n-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon, *Inf. Process. Lett.*, 114(10):515-518pp.

Chuzhoy, J. and Naor, J.S., 2002, Covering problems with hard capacities, in the 43rd Annual IEEE Symposium on Foundations of Computer Science, 481-489pp.

Clark, B.N., Colbourn, C.J., and Johnson, D.S., 1990, Unit disk graphs, *Discrete Mathematics*, 86(1):165-177pp.

Cygan, M., Pilipczuk, M., and Wojtaszczyk, J.O., 2011, Capacitated domination faster than o(2n), *Inf. Proc. Let.*, 111:1099-1103pp.

Datta, A.K., Gurumurthy, S., Petit, F., and Villain, V., 2000, Self-stabilizing network orientation algorithms in arbitrary rooted networks, 20th IEEE International Conference on Distributed Computing Systems, 576-583pp.

Datta, A.K., Larmore, L.L., Devismes, S., Heurtefeux, K., and Rivierre, Y., 2013, Self-stabilizing small k-dominating sets, *International Journal of Networking and Computing*, 3(1): 113-116pp.

Dhawan, A., Tanco, M., and Scoville, N., 2014, A Distributed Greedy Algorithm for Constructing Connected Dominating Sets in Wireless Sensor Networks, Proceedings of the 3rd International Conference on Sensor Networks, 1:181-187pp.

Dijkstra, E.W., 1974, Self-stabilizing systems in spite of distributed control, *Commun. ACM*, 17:643-644pp.

Dolev, S., 2000, Self-stabilization, Cambridge, MA, USA: MIT Press, 208p.

Dom, M., Lokshtanov, D., Saurabh, S., and Villanger, Y., 2008, Capacitated domination and covering: A parameterized perspective, in Parameterized and Exact Computation, Heidelberg: Springer, 78-90pp.

Dubois, S. and Tixeuil, S., 2011, A taxonomy of demons in self-stabilization, CoRR.

Ephremides, E., Wieselthier, J. E., and Baker, D. J., 1987, A design concept for reliable mobile radio networks with frequency hopping signaling, Proceedings of the IEEE, 75(1):56-73pp.

Erciyes, K., 2013, Distributed Graph Algorithms for Computer Networks, Springer Publishing Company, Incorporated, 328p.

Fischer, M. and Noever, A., 2018, Tight analysis of parallel randomized greedy mis, in Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’18, (Philadelphia, PA, USA), 2152-2160pp.

Funke, S., Kesselman, A., Meyer, U., and Segal, M., 2006, A simple improved distributed algorithm for minimum cds in unit disk graphs, *ACM Trans. Sen. Netw.*, 2(3):444-453pp.

Gandhi, R., Halperin, E., Khuller, S., Kortsarz, G., and Srinivasan, A., 2006, An improved approximation algorithm for vertex cover with hard capacities, *Journal of Computer and System Sciences*, 72(1):16-33pp.

Gao, B., Yang, Y., and Ma, H., 2005, A new distributed approximation algorithm for constructing minimum connected dominating set in wireless ad hoc networks, *International Journal of Communication Systems*,18(8):743-762pp.

Gao, S. and Zhang, Y., 2012, An improved distributed approximation algorithm for minimum connected dominating set, 8th International Conference on Natural Computation, 1019-1022pp.

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

Ghaffari, M., 2017, Distributed mis via all-to-all communication, in Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC ’17, (New York, NY, USA), 141-149pp.

Goddard, W. and Srimani, P.K., 2010, Anonymous self-stabilizing distributed algorithms for connected dominating set in a network graph.

Goddard, W., Hedetniemi, S., Jacobs, D., and Srimani, P., 2003, Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks, Parallel and Distributed Processing Symposium, 14pp.

Goldberg, M. and Spencer, T., 1987, A new parallel algorithm for the maximal independent set problem, in 28th Annual Symposium on Foundations of Computer Science, 161-165pp.

Guellati, N. and Kheddouci, H., 2010, A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs, *J. Parallel Distrib. Comput.*, 70:406-415pp.

Guha, S. and Khuller, S., 1998, Approximation algorithms for connected dominating sets, *Algorithmica*, 20(4).

Guha, S., Hassin, R., Khuller, S., and Or, E., 2003, Capacitated vertex covering, *Journal of Algorithms*, 48(1):257-270pp.

Hedetniemi, S.M., Hedetniemi, S.T., Jacobs, D.P., and. Srimani, P.K., 2003, Self-stabilizing algorithms for minimal dominating sets and maximal independent sets, *Computers & Mathematics with Applications*, 46(5-6):805-811pp.

Henry, E. and Adamchuk, V., Stanhope, T., Buddle, C. and Rindlaub, N., 2019, Precision apiculture: Development of a wireless ad hoc and sensor network for honeybee hives, *Computers and Electronics in Agriculture*, 156:138-144pp.

Herman S., 2000, Phase clocks for transient fault repair, IEEE Transactions on Parallel and Distributed Systems, 11(10):1048-1057pp.

Herman, T.R., 1992, Adaptivity Through Distributed Convergence, The University of Texas at Austin, USA.

Ikeda, M., Kamei, S., and Kakugawa, H., 2002, A space-optimal self-stabilizing algorithm for the maximal independent set problem, in Proceedings 3rd International Conference on Parallel and Distributed Computing, Applications and Technologies.

Jain, A. and Gupta, A., 2005, A distributed self-stabilizing algorithm for finding a connected dominating set in a graph, in Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PD-CAT'05), 615-619pp.

Jain, B., Brar, G., Malhotra, J., and Rani, S., 2017, A novel approach for smart cities in convergence to wireless ad hoc and sensor networks, Sustainable Cities and Society, 35:440-448pp.

Jallu, R.K, Prasad, P.R., and Das, G.K., 2017, Distributed construction of connected dominating set in unit disk graphs, *Journal of Parallel and Distributed Computing*, 104:150-166pp.

Kamei, S. and Kakugawa, H., 2007, A self-stabilizing distributed approximation algorithm for the minimum connected dominating set, in 2007 IEEE International Parallel and Distributed Processing Symposium, 1(8):613pp.

Kamei, S. and Kakugawa, H., 2008, A self-stabilizing approximation for the minimum connected dominating set with safe convergence, in T. P. Baker,

A. Bui, S. Tixeuil (Eds.), Principles of Distributed Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 496-511pp.

Kamei, S. and Kakugawa, H., 2012, A self-stabilizing 6-approximation for the minimum connected dominating set with safe convergence in unit disk graphs, *Theor. Comput. Sci.*, 428:80-90pp.

Kamei, S., Izumi, T., and Yamauchi, Y., 2016, An asynchronous self-stabilizing approximation for the minimum cds with safe convergence in udgs, *Theoretical Computer Science*, 615:102-119pp.

Kao, M.J., and Chen, H.L., 2010, Approximation Algorithms for the Capacitated Domination Problem, *Frontiers in Algorithmics*, 185-196pp.

Kao, M.J. and Liao, C.S., 2007, Capacitated Domination Problem, *Algorithms and Computation*, 256-267pp.

Kao, M.J., Chen, H.L., and Lee, D.T., 2015, Capacitated domination: Problem complexity and approximation algorithms, *Algorithmica*, 72:1-43pp.

Kao, M.J., Liao, C.S., and Lee, D.T., 2011, Capacitated domination problem, *Algorithmica*, 60:274-300pp.

Kao, M.J., Shiau, J.Y., Lin, C.C., and Lee, D., 2019, Tight approximation for partial vertex cover with hard capacities, *Theoretical Computer Science*.

Karp, R.M. and Wigderson, A., 1985, A fast parallel algorithm for the maximal independent set problem, *J. ACM*, 32:762-773pp.

Khuller, S., Purohit, M., and Sarpatwar, K.K., 2014, Analyzing the optimal neighborhood: Algorithms for budgeted and partial connected dominating set problems, in: Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '14, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1702-1713pp.

Kim, D., Wu, Y. Li, Y., Zou, F., and Du, D., 2009, Constructing minimum connected dominating sets with bounded diameters in wireless networks, IEEE Transactions on Parallel and Distributed Systems, 20(2):147-157pp.

Kuhn, F. and Moscibroda, T., 2010, Distributed approximation of capacitated dominating sets, *Theory of Computing Systems*, 47:811-536pp.

Kuhn, F., Moscibroda, T., Nieberg, T., and Wattenhofer, R., 2005, Fast deterministic distributed maximal independent set computation on growth-bounded graphs, in DISC.

Li, R., Hu, S., Zhao, P., Zhou, Y., and Yin, M., 2017, A novel local search algorithm for the minimum capacitated dominating set, *Journal of the Operational Research Society*.

Li, Y., Thai, M.T., Wang, F., Yi, C.W., Wan, P.J., and Du, D.Z., 2005, On greedy construction of connected dominating sets in wireless networks: Research articles, *Wirel. Commun. Mob. Comput.*, 5(8):927-932pp.

Liedloff, M., Todinca, I., and Villanger, Y., 2014, Solving capacitated dominating set by using covering by subsets and maximum matching, *Discrete Applied Mathematics*, 168:60-68pp.

Lin, J.C. and Huang, T.C., 2003, An efficient fault-containing self-stabilizing algorithm for finding a maximal independent set, IEEE Transactions on Parallel and Distributed Systems, 14(8):742-754pp.

Linial, N., 1992, Locality in distributed graph algorithms, *SIAM J. Comput.*, 21:193-201pp.

Liu, Z. Wang, B., and Guo, L., 2010, A survey on connected dominating set construction algorithm for wireless ad hoc and sensor networks, *Information Technology Journal*, 9:1071-1092pp.

Luby, M., 1986, A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, 15:1036-1055pp.

Luczak, T. and Szymaska, E., 1997, A parallel randomized algorithm for finding a maximal independent set in a linear hypergraph, *Journal of Algorithms*, 25(2):311-320pp.

Luo, C., Yu, J., Li, D., Chen, H., Hong, Y., and Ni, L., 2018, A novel distributed algorithm for constructing virtual backbones in wireless ad hoc and sensor networks, *Computer Networks*, 146:104-114pp.

Marathe, M.V., Breu, H., Hunt H.B, Ravi, S.S., and Rosenkrantz D.J., 1995, Simple heuristics for unit disk graphs, *Networks*, 25(2):59-68pp.

MEMSIC. Inc, Iris datasheet, http://www.memsic.com/userfiles/files/Datasheets/WASN/IRIS_Datasheet.pdf (Erişim tarihi: 30.05.2019).

Min, M., Du, H., Jia, X., Huang, C.X, Huang, S.C.H., and Wu, W., 2006, Improving construction for connected dominating set with Steiner tree in wireless ad hoc and sensor networks, *Journal of Global Optimization*, 35(1):111-119pp.

Mohanty, J.P., Mandal, C., and Reade, C., 2017, Distributed construction of minimum connected dominating set in wireless ad hoc and sensor network using two-hop information, *Computer Networks*, 123:137-152pp.

Potluri, A. and Singh, A., 2013, Metaheuristic algorithms for computing capacitated dominating set with uniform and variable capacities, *Swarm and Evolutionary Computation*, 13:22-33pp.

Pradhan, D., 2012, Algorithmic aspects of k-tuple total domination in graphs, *Information Processing Letters*, 112(21):816-822pp.

Raei, H., Sarram, M., and Adibniya, F., 2008, Distributed algorithm for connected dominating sets in wireless ad hoc and sensor networks with different transmission ranges, in 2008 International Symposium on Telecommunications, 337-342pp.

Ramson, S.R.J. and Moni, D. J., 2017, Applications of wireless ad hoc and sensor networks a survey, 2017 International Conference on Innovations in Electrical, Electronics, Instrumentation and Media Technology (ICEEIMT), 325-329pp.

Rashid, B. and Rehmani, M.H., 2016, Applications of wireless ad hoc and sensor networks for urban areas: a survey, *Journal of Network and Computer Applications*, 60:192-219pp.

Rostami, A.S., Badkoobe, M., Mohanna, F., Keshavarz, H., Hosseinabadi, A.A.R., and Sangaiah, A.K., 2018, Survey on clustering in heterogeneous and homogeneous wireless ad hoc and sensor networks, *The Journal of Supercomputing*, 74:277-323pp.

Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., and Ko, K.I., 2004, A greedy approximation for minimum connected dominating sets, *Theoretical Computer Science*, 329(1):325-330pp.

Schneider, J. and Wattenhofer, R., 2008, A log-star distributed maximal independent set algorithm for growth-bounded graphs, in Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing, PODC ’08, (New York, NY, USA), 35-44pp.

Scott, A., Jeavons, P., and Xu, L., 2013, Feedback from nature: An optimal distributed algorithm for maximal independent set selection, in Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing, PODC ’13, (New York, NY, USA),147-156pp.

Shang, W. and Wang, X., 2011, Algorithms for minimum connected capacitated dominating set problem, Discrete Mathematics, *Algorithms and Applications*, 3(1):9-15pp.

Shi, Z., Goddard, W., and Hedetniemi, S.T., 2004, An anonymous self-stabilizing algorithm for 1-maximal independent set in trees, *Information Processing Letters*, 91(2):77-83pp.

Shukla, S.K., Rosenkrantz, D.J., Ravi, S.S., and Shukla, E. K., 1995, Observations on self- stabilizing graph algorithms for anonymous networks (extended abstract), in Proceedings of the Second Workshop on Self-Stabilizing Systems,1-7pp.

Siegemund, G., 2017, Self-stabilizing algorithms in wireless ad hoc and sensor networks, PhD dissertation, Hamburg University of Technology, Hamburg, Germany.

Singh, S.P. and Sharma, S., 2015, A survey on cluster-based routing protocols in wireless ad hoc and sensor networks, Procedia Computer Science, International Conference on Advanced Computing Technologies and Applications (ICACTA), 45:687-695pp.

Surendran, S. and Vijayan, S., 2015, Distributed computation of connected dominating set for multi-hop wireless networks, *Procedia Computer Science*, 63:482-487pp.

Tel, G., 2001, Introduction to Distributed Algorithms, Cambridge University, 2nd edition, 612p.

Thai, M.T. and Du, D.Z., 2006, Connected dominating sets in disk graphs with bidirectional links, *IEEE Communications Letters*, 10(3):138-140pp.

Tixeuil, S., 2009, Algorithms and Theory of Computation Handbook, Second Edition, chapter Self-stabilizing Algorithms, CRC Press, 9-10pp.

Turau, V., 2007, Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler, *Information Processing Letters*, 103(3):88-93pp.

Vijayasharmila, S.K.S., Kumar, P.G., and Kamalesh, S., 2015, A survey on connected dominating sets (cds) both in the wireless ad hoc and sensor networks and wireless ad hoc networks, International journal of engineering research and technology (IJERT), 04(02).

Vinayagam, P.S., 2016, A survey of connected dominating set algorithms for virtual backbone construction in ad hoc networks, International Journal of Computer Applications, 143(9):30-36pp.

Wan, P.J., Alzoubi, K.M., and Frieder, O., 2004, Distributed construction of connected dominating set in wireless ad hoc networks, *Mobile Networks and Applications*, 9(2):141-149pp.

Yang, W.H., Wang Y.C., and Tseng, Y.C., 2012, Efficient packet recovery using prioritized network coding in dvb-ipdc systems, IEEE Communications Letters, 16(3):382-385pp.

Yu, J., Wang, N., Wang, G., and Yu, D., 2013, Review: Connected dominating sets in wireless ad hoc and sensor networks - a comprehensive survey, *Comput. Commun.*, 36 (2):121-134pp.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Orhan Dağdeviren for his significant guidance, support, patience, friendly behavior, since it provides me with an excellent atmosphere which contains free working environment through my scientific research. Also, I thank him for letting me a practical experience beyond the textbooks, patiently corrected my writing papers and dissertations since my MSc.

I would also like to thank the members of the thesis monitoring committee, Prof. Dr. Mehmet Emin DALKILIÇ and Assoc. Prof. Dr. Hasan BULUT for their fruitful discussions and constructive feedbacks.

I would like to thank TUBITAK (Scientific and Technical Research Council of Turkey) for supporting my work with the project grant (Project number 215E115).

Special thanks go to my mother, father, and brothers for their encouraging me with their best wishes and prayers.

I would also deeply thank my wife for her supporting me and my son for his continuous moral support by his existence.

This dissertation is dedicated to my son Tuğkan Arapoğlu and my wife Tuğba Arapoğlu who are meaning of my life.

CURRICULUM VITAE

Özkan ARAPOĞLU

Address: Ege University International Computer Institute, Izmir/TURKEY

E-mail: ozkanarapogu@hotmail.com, ozkanarapoglu@gmail.com

Nationality: Turkish

Birth Place and Date: Çorum, 29.12.1985

Research Interests: Distributed Algorithms, Graph Theory, Wireless Networks, IoT, Quantum Programming, Artificial Intelligence, Computer Vision

Education

PhD: 2015-2019, Information Technology, International Computer Institute, Ege University, Izmir/Turkey

MSc: 2012-2015, Information Technology, International Computer Institute, Ege University, Izmir/Turkey

BSc: 2015-2019, Computer Engineering, Khoja Akhmet Yassawi International Kazakh-Turkish University, Ankara/Turkey

BSc: 2004-2008, Computer Education and Instructional Technology, Faculty of Necatibey Education, Balıkesir University, Balıkesir/Turkey

Work Experiences

2007-2008: Software Developer, Web Team, Beşiktaş/İstanbul

2008-2014: Information Technology Teacher, Kozağaç Elementary School, Ministry of National Education, Izmir/Turkey

2014-2015: Information Technology Teacher, Şehit Üsteğmen Konuralp Özcan Elementary School, Ministry of National Education, Izmir/Turkey

2015- : Information Technology Teacher, Konak Şehit Ömer Halisdemir Science and Art Center, Ministry of National Education, Izmir/Turkey

Foreign Languages

Turkish (Native)

English: Advanced

Project

2016-2018: Distributed and self-stabilizing capacitated graph theoretical algorithms: Funded by TUBITAK (215E115). Role: Researcher

Publications

National Conference:

Arapoglu, O. and Dagdeviren O., 2017, Energy-Efficient Distributed Self Stabilizing Maximal Independent Set Algorithms for Wireless Sensor Networks, 2017 Akademik Bilişim Konferansı. (Published)

International Conferences (IEEE):

Akram, V.K., Arapoglu, O., and Dagdeviren, O., 2018, A Depth-First Search based Connectivity Estimation Approach for Fault Tolerant Wireless Sensor Networks, IEEE 2018 International Conference on Artificial Intelligence and Data Processing (IDAP). (Published)

Evcimen, H.T., Arapoglu, O., and Dagdeviren, O., 2018, SELFSIM: A Discrete-Event Simulator for Distributed Self-Stabilizing Algorithms, IEEE 2018 International Conference on Artificial Intelligence and Data Processing (IDAP). (Published)

Arapoglu, O. and Dagdeviren, O., 2019, An Asynchronous Self-Stabilizing Maximal Independent Set Algorithm In Wireless Sensor Networks Using Two-Hop Information, 2019 International Symposium on Networks, Computers and Communications (ISNCC). (Presented)

Arapoglu, O. and Dagdeviren, O., 2019, A Fully Distributed Fault-Tolerant Cluster Head Selection Algorithm for Unit Disk Graphs, 2019 International Symposium on Networks, Computers and Communications (ISNCC). (Presented)

Journal Papers (SCIE):

Arapoglu, O., Akram, V.K., and Dagdeviren, O., 2019, An energy-efficient, self-stabilizing and distributed algorithm for maximal independent set construction in wireless ad hoc and sensor networks," Computer Standards and Interfaces, 62:32-42pp. (Published)

Arapoglu, O. and Dagdeviren, O., 2019, Fault-tolerant capacitated maximal independent set construction in wireless ad hoc and sensor networks, Springer Wireless Networks. (Under Review)

Arapoglu, O. and Dagdeviren, O., 2019, Distributed fault-tolerant minimum capacitated dominating set construction in wireless ad hoc and sensor networks, IEEE Sytems Journal. (Under Review)

Arapoglu, O. and Dagdeviren, O., 2019, A fault-tolerant and distributed capacitated connected dominating set algorithm for wireless ad hoc and sensor networks, Elsevier Computer Standards and Interfaces. (Under Review)

Book Chapter:

Ileri, C.U., Yigit, Y., Arapoglu, O., Evcimen, H.T., Asci, M., and Dagdeviren, O., 2019, Capacitated Graph Theoretical Algorithms for Wireless Sensor Networks towards Internet of Things, Handbook of Research on the IoT, Cloud Computing, and Wireless Network Optimization, IGI Global, 563p. (Published)