République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A.MIRA-BEJAIA

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Faculté des Sciences Exactes
Département d'Informatique

THÈSE

Présentée par

LALOUCI Ali

Pour l'obtention du grade de

DOCTEUR EN SCIENCES

Filière : Informatique

Option : Réseaux et Systèmes Distribués

Thème

Toward a distributed approach for IoT network protection

Soutenue le : 28/06/2025

Devant le Jury composé de :

Nom et Prénom	Grade		
Mr AMROUN Kamal	Professeur	Univ. de Bejaia	Président
Mr FARAH Zoubeyr	MCA	Univ. de Bejaia	Rapporteur
Mr AZOUAOU Faical	Professeur	ESTIN. de Bejaia	Examinateur
Mr BOUKRAA Doulkifli	Professeur	Univ. de Jijel	Examinateur
Mr AHMIA Moussa	Professeur	Univ. de Jijel	Examinateur
Mr ALIOUA Ahmed	MCA	Univ. de Jijel	Examinateur

Année Universitaire : 2024/2025.

Acknowledgements

First and foremost, I thank Allah Almighty for giving me the strength, the courage and the knowledge to complete my doctoral degree.

I would like to express my deepest gratitude to my supervisor Research Director Zoubeyr Farah, for his continuous support and guidance, monitoring, orientations and relevant remarks, which steered me in the right direction to complete my work.

I would like to express my sincere gratitude to Professor Volker Turau, Director of the Institute of Telematics at Hamburg University of Technology, for his warm welcome, invaluable guidance, and support throughout my internship.

I would like to express my sincere gratitude to Mehmet Gümüş, Professor at Zonguldak Bülent Ecevit University, for his warm welcome, invaluable guidance, and support throughout my internship.

I am very grateful to the members of my dissertation committee, Professor AMROUN Kamal, Professor AHMIA Moussa, Professor AZOUAOU Faical, Professor BOUKRAA Doulkifli, and Doctor ALIOUA Ahmed for their acceptance evaluating my work and for the time they spent to review my thesis manuscript.

I would like to express my sincere thanks to the entire staff of the Computer Science Department of University of Bejaia and Mila University Center, for the quality of their training, the social and scientific atmosphere, and the administrative responsiveness they have shown over the years.

I would like to express my heartfelt gratitude to my parents for their unwavering efforts and sacrifices throughout my life. A special thanks to my wife, my son, and my daughter for their understanding and patience throughout the years of this work. I also want to

acknowledge my brothers and sisters, whose constant support and encouragement were invaluable during the preparation of this thesis.

I would like to warmly thank my friends and colleagues, and to ensure no one is forgotten, I extend my gratitude to all those, near or far, who contributed with their advice, encouragement, or friendship to the outcome of this modest work. Here, I express my deep appreciation.

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

The Internet is a vast global network that is progressively evolving into a large-scale infrastructure known as the *Internet of Things (IoT)*, seamlessly connecting billions of individuals and tens of billions of devices. The concept of the IoT was established first in 1999 by Kevin Ashton, a British technology pioneer, during his work at Procter and Gamble[1]. He coined the term while working on a project to integrate radio-frequency identification (RFID) with the internet, to track the flow of goods in a supply chain. Ashton envisioned a world where everyday objects would be connected to the internet, allowing for the exchange of data and creating smart systems [2]. The IoT enables smart devices to communicate seamlessly with one another and with other internet-connected systems. By integrating components such as smartphones, sensors, and gateways, it forms a vast network of interconnected devices capable of exchanging data and autonomously performing a wide range of tasks. This network of interlinked devices, frequently constrained in processing capacity and security features, is increasingly susceptible to diverse cyber threats and vulnerabilities. The inherent challenges in IoT security, such as limited device resources, decentralized architecture, and the demand for real-time responses, underscore the necessity of efficient, scalable, and secure frameworks to safeguard sensitive data and maintain robust network integrity.

Securing an IoT system is a complex and demanding task, particularly in large-scale, heterogeneous, and resource-constrained environments. Traditional security mechanisms are often unsuitable for IoT devices due to their limited computational and energy resources. Consequently, novel security frameworks must be developed to ensure authentication, confidentiality, and data integrity while remaining compatible with IoT constraints. In this context, several emerging technologies and approaches such as advanced cryptography, intrusion detection system, blockchain, fog computing, machine learning, and graph parameters have been introduced to enhance IoT security and resilience.

In response to this challenge, this thesis investigates the application of graph theory as a powerful framework for modeling and mitigating security vulnerabilities in IoT *networks*.

[1] https://us.pg.com/

Graph theory offers a systematic approach to analyzing the relationships and interactions among network nodes, enabling the evaluation of key graph parameters that enhance network resilience and security. A common approach in solving problems using graphs consists of the following steps: i) modeling the problem by a graph, ii) finding the required parameter corresponding to the studied problem, and iii) developing efficient algorithms to determine the value of this parameter.

Critical parameters such as the *dominating nodes set*, *secure dominating nodes set*, *maximal independent nodes set*, and *critical nodes set* are commonly used to identify key nodes, optimize network performance, and strengthen defenses against failures and cyber threats. Traditional sequential algorithms for computing these graph parameters are impractical for *IoT networks* due to their exponential time complexity in obtaining optimal solutions. Moreover, they are inadequate for implementation on recent distributed systems and architectures. Given the NP-complete nature of determining exact values for these parameters, this thesis focuses on the development and evaluation of heuristic and distributed algorithms that efficiently approximate near-optimal solutions while significantly reducing computational overhead.

This thesis aims towards developing a distributed approach using graph parameters to improve the security and resilience of *IoT networks*. The study focuses on three key graph-theoretic problems: the *Secure Dominating Set Problem (SDSP)*, *Maximal Independent Set Problem (MISP)*, and *Critical Node Detection Problem (CNDP)*, which play a crucial role in optimizing *network protection*. To evaluate and validate the proposed approach, simulation technique will be employed. Specifically, the *CupCarbon IoT* simulator[2], a widely used tool for designing *IoT networks* and testing *distributed algorithms*, will be utilized to assess the effectiveness and scalability of the proposed solutions.

This thesis is organized as follows:

Chapter 1 provides an overview of fundamental concepts related to *IoT networks*, including their architecture, core components, and inherent security challenges. It explores various *IoT* security techniques and emerging solutions, such as cryptography, machine learning, and blockchain. Additionally, it introduces graph-theoretic approaches and key parameters that underpin the security methodologies proposed in this research. Furthermore, the chapter presents the fundamental principles of distributed systems, including communication models and the formal definition of *distributed algorithms*. Finally, it examines various distributed computing platforms, highlighting their key characteristics and relevance to the proposed security framework.

[2]https://cupcarbon.com/

In Chapter 2, we address the *Secure Dominating Set Problem (SDSP)* in *IoT networks*, a crucial variant of the *Dominating Set Problem (DSP)* that incorporates security constraints. To tackle the *SDSP*, this chapter provides a formal problem statement, a review of relevant literature, and the theoretical foundations. We introduce distributed heuristic-based algorithm designed to solve this problem and analyze their expected performance.

In Chapter 3, we investigate the second parameter, the *Maximal Independent Set (MIS)*. We present the theoretical foundations of the *Maximal Independent Set Problem (MISP)* and provide a comprehensive review of existing *distributed algorithms*, both deterministic and randomized, for solving it in distributed environments. Building upon this analysis, we propose a novel *distributed algorithm* specifically designed for *IoT networks* and conduct a thorough evaluation of its computational complexity and convergence properties.

In Chapter 4, we examine the *Critical Node Detection Problem (CNDP)* in tree-based *IoT networks*, focusing on the identification of nodes whose elimination would critically affect network connectivity and security. To address this challenge, we propose a *distributed algorithm* for detecting *critical nodes* in *IoT network* structures and conduct a comprehensive analysis of its expected performance and efficiency.

In Chapter 5, we conduct a simulation-based performance analysis of the proposed algorithms that comprise our approach, evaluating them under diverse *IoT* configurations, including various network topologies and security threat models. The experimental results are presented and analyzed in comparison with existing solutions, highlighting the improvements in efficiency, scalability, and robustness achieved by our methods.

In Conclusion, we summarize the key findings and contributions of this thesis, discussing their implications for the field of *IoT* security. We also outline limitations and propose directions for future research, including potential extensions of the algorithms and frameworks developed, as well as applications in other domains within distributed networks.

Chapter 1

Background: IoT Network Protection, Graph Parameters, and Distributed Computing

1.1 Introduction

Chapter 1 provides a comprehensive overview of the foundational concepts essential to this research. It begins by underscoring the importance of *IoT networks*, examining their architecture, components, and diverse applications. The chapter further presents the security challenges inherent to *IoT* systems. Additionally, it explores emerging security techniques, emphasizing their pivotal role in addressing these vulnerabilities and enhancing the protection of *IoT* infrastructures.

Furthermore, the chapter introduces graph theory as a powerful tool for modeling and addressing *IoT network* vulnerabilities, with a focus on key parameters such as the *domination set, maximal independent set, and critical node set.* These parameters are essential for enhancing network resilience and protection. The discussion also extends to the principles of distributed computing, examining distributed systems, algorithms, and platforms, which collectively form a theoretical framework for securing *IoT networks* and guide the contributions presented in subsequent chapters.

1.2 IoT basic concepts

This first section provides a comprehensive introduction to the concepts of basic principles of the *IoT*. It begins with a concise definition of *IoT*, highlighting its core principles and characteristics. Following this, the section presents the architecture of *IoT*, discussing its

key components, layers, and their interconnections. Finally, it explores the diverse application domains of *IoT*, emphasizing its transformative impact across various industries and sectors. Through this structured overview, readers will gain a clear understanding of *IoT*'s scope, framework, and real-world relevance.

1.2.1 Definition of IoT

The concept of the *Internet of Things (IoT)* was first introduced by Kevin Ashton in 1999 [3]. The objective of the *IoT* is to broaden internet connectivity beyond traditional digital devices (computers, smartphones, and tablets) to everyday objects and appliances, including simple devices. By embedding connectivity, data collection, and processing capabilities, the *IoT* transforms these devices smart objects, enhancing functionality and enabling more informed, data-driven decisions. Leveraging advancements in artificial intelligence and networked communication, *IoT* enriches numerous aspects of personal, professional, and industrial life, fostering greater efficiency, automation, and interactivity across a wide range of applications.

The *Internet of Things* is a global network that links various devices, including common objects that possess network interface cards, computational abilities, and storage capacity [4]. Numerous communication networks connect devices, including RFID (Radio Frequency Identification), ZigBee, WSN, WLAN (Wireless Local Area Network), NFC (Near Field Communication), DSL (Digital Subscriber Line), GPRS (General Packet Radio Service), LTE (Long Term Evolution), LoRaWAN (Long Range Wide Area Network), Bluetooth, and 4G/5G. For more details on *IoT* systems, refer to [5]. Figure 1.1 illustrates example devices commonly found in an *IoT* system.

Figure 1.1: IoT system.

1.2.2 IoT architecture

The architecture of the *IoT* remains largely non-standardized, though a typical *IoT* architecture is generally composed of three distinct layers: perception, network, and application [6], as depicted in Figure 1.2.

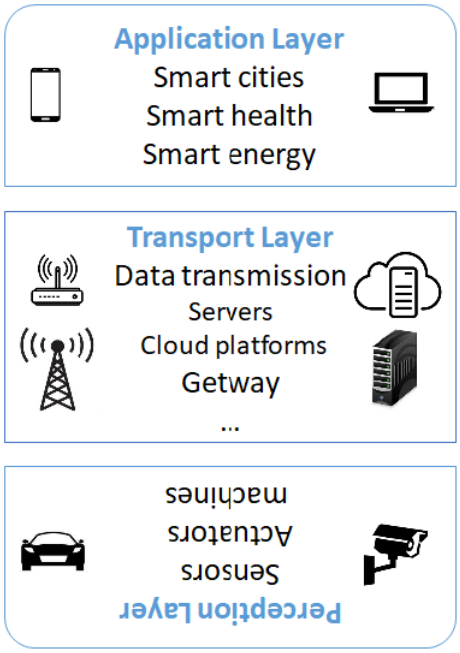


Figure 1.2: IoT architecture

- **Perception Layer (Physical or Sensor):** the Physical layer encompasses *IoT* devices, including sensors, actuators, and machines, which are responsible for sensing, processing, and communicating with other devices. Sensors in this layer detect physical changes in the environment and gather relevant data. Often referred to as the *client side*, this layer operates at the client's location, where it collects information to be passed on to the network layer, the aggregation or *IoT* gateway layer. In the aggregation layer, data is consolidated and processed, often involving servers managed by operators.

- **Transport Layer (Connectivity):** the transport layer is responsible for transferring data between *IoT* devices and servers or cloud platforms. It relays raw data from sensors and *IoT* devices to central systems and sends processed responses back to the devices. Data transmission occurs through networks or gateways, utilizing a range of connectivity technologies such as Wi-Fi, Ethernet, Zigbee, Bluetooth, LoRa, and cellular networks.

- **Application Layer (API Management):** the final layer, known as the API

management or application layer, serves as the interface between third-party applications and the underlying IoT infrastructure or end-users. This layer enables secure interactions and data exchange, supported by device management, identity, and access management systems that ensure safety and security throughout the entire IoT ecosystem.

1.2.3 IoT applications

The IoT provides numerous applications that improve different facets of daily life and activities. Significant domains of IoT implementation encompass [7, 8]:

- **Smart homes and cities:** a smart home integrates a range of interconnected devices, such as smart clocks, baby monitors, and fire detectors, that communicate over local wireless networks to automate and enhance home environments. Meanwhile, a smart city leverages technology on a larger scale, incorporating an extensive network of sensors and connected devices throughout urban areas. These devices continuously gather and exchange real-time data to optimize public services, improve energy efficiency, and enhance the quality of life for residents.

- **Smart healthcare:** smart healthcare systems facilitate the collection, transmission, and storage of patients' physiological data, enabling continuous health monitoring and timely medical interventions. For instance, wearable medical sensors can monitor a patient's heart rate, transmitting real-time data to hospital servers. This enables healthcare providers to track, diagnose, and respond to health conditions with enhanced precision and efficiency.

- **Smart agriculture:** for decades, large-scale industrialized agriculture has expanded to meet the demands of growing populations and ensure global food security. Smart agriculture leverages technology to remotely monitor and control critical factors such as temperature, humidity, irrigation, soil moisture, and microclimate conditions, maximizing crop yield and quality while minimizing financial risks. In intelligent farming systems, sensors can also be attached to livestock to monitor animal behavior and health, allowing for early detection of issues and more efficient management of livestock well-being.

- **E-Commerce and trading:** although this segment has attracted the most criticism regarding connected devices, IoT has significantly enhanced the targeting of advertising campaigns. By collecting behavioral data and leveraging machine and

deep learning for profiling, *IoT* enables highly personalized marketing strategies that cater to individual preferences and purchasing behaviors.

- **Smart Industry:** also known as Industrial *IoT* (IIoT), smart industry leverages machine-to-machine (M2M) communication to automate manufacturing processes with minimal human intervention. IIoT enhances control over production, data management, and issue resolution, ultimately delivering more efficient, reliable, and high-quality products.

1.3 IoT networks security

This section offers an overview of the key concepts underlying *IoT security*. It begins with a concise definition of *IoT security*, outlining its importance in safeguarding connected systems. Following this, the key objectives of *IoT security* are discussed, emphasizing the critical aspects that ensure system resilience and data integrity. Subsequently, the section examines various types of attacks that target *IoT* systems, shedding light on the vulnerabilities and challenges posed by these threats. Finally, it reviews emerging security solutions documented in the literature, highlighting the innovative approaches and strategies developed to protect *IoT* systems effectively.

1.3.1 The principles of security

In computer science, security refers to a comprehensive set of measures, policies, and technologies aimed at protecting computer systems, networks, and data from unauthorized access, misuse, disruption, modification, or destruction. The primary objective of security is to defend against threats such as cyberattacks, data breaches, and system vulnerabilities, thereby ensuring the reliability, integrity, and trustworthiness of digital systems. Security is underpinned by the following core principles [9, 10]:

- **Privacy and confidentiality:** the most crucial aspect of network security is privacy. Confidentiality ensures the privacy of data transmission between a sender and a recipient by guaranteeing that access to the data is restricted to authorized entities. The most effective method to ensure data confidentiality is through cryptography or data encryption.

- **Authentication and access control:** authentication is the process used to verify a node's identity before it can communicate with other nodes. A sensor node must validate the identity of the source node to prevent attackers from forging and injecting fake packets into the network.

- **Integrity:** integrity ensures that data remains accurate, consistent, and trustworthy throughout its lifecycle. It protects information from unauthorized modification, deletion, or tampering, either during transmission or while stored. The primary goal of integrity is to prevent data from being altered in ways that compromise its authenticity or usability.

- **Availability:** due to a significant proportion of network nodes relying on battery power, ensuring the continuous availability of services in the IoT remains challenging. Techniques facilitating an automatic conversion to sleep mode throughout idle intervals enhance traditional methods while optimizing energy efficiency.

1.3.2 Classification of attacks

IoT attacks can be broadly categorized based on various criteria such as layers, objectives, and countermeasures, as outlined in [7]. In our thesis, we present a taxonomy of IoT attacks organized by layers, as illustrated in Figure 1.3, and inspired from [7, 11–13].

- **Perception/Sensing Layer:** due to the nature of IoT, some devices may be deployed in remote, unmonitored areas, making them vulnerable to physical tampering. Attackers with direct access to a device could damage it, attempt brute-force attacks to retrieve cryptographic keys, or modify its firmware to execute unauthorized actions within the network. Common examples of attacks at the perception layer include side-channel attacks, botnet attacks (e.g., Mirai), and other forms of hardware and firmware exploitation.

- **Network layer:** attackers can exploit vulnerabilities within the network stack to perform various malicious activities. These include executing man-in-the-middle attacks by poisoning network services, manipulating devices to act as sinkholes to disrupt data flow, or passively sniffing network traffic to collect data for future attacks. Common examples of attacks at the network layer include DoS attacks, data-in-transit attacks, and spoofing attacks.

- **Application/service layer:** applications are not immune to security threats. At this layer, attackers can exploit vulnerabilities stemming from poor programming practices, such as inadequate input validation or improper error handling, to inject malicious scripts or code. Additionally, attackers may leverage users' lack of awareness regarding security best practices, which makes social engineering attacks particularly effective at this level. Addressing these risks requires robust coding standards, regular security audits, and user education to mitigate threats effectively.

Common examples of attacks at the application layer include phishing, sniffing, and trust management attacks.

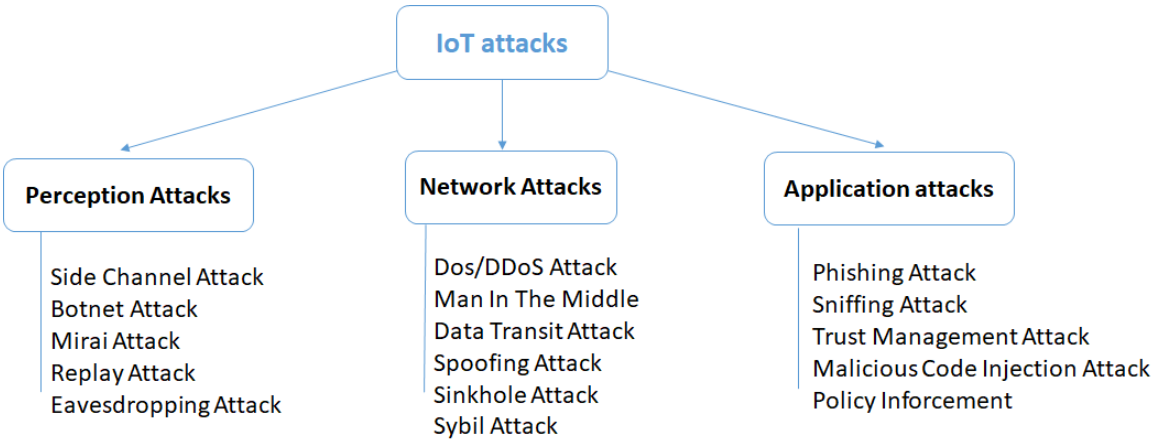


Figure 1.3: Common attacks against *IoT* layers.

1.3.3 Emerging solutions for security

In the literature, several solutions have been proposed to enhance the security of *IoT networks*. Among the most well-known are those cited in [7, 14], as illustrated in Figure 1.4:

- **Cryptography:** cryptography is a set of techniques used to transform information from its original, intelligible form, known as *plaintext*, into an unintelligible, coded form, known as *ciphertext*. This process is called encryption. In symmetric encryption, a single key is used for both encryption and decryption. In contrast, asymmetric encryption employs two keys: a private key and a public key. Cryptography is an effective tool to ensure confidentiality, integrity, and authentication of data.

  However, most *IoT* devices face significant challenges due to their limited processing power, memory, and battery capacity. Consequently, traditional cryptographic algorithms are often unsuitable for these resource-constrained devices. To address this, advanced cryptographic techniques such as lightweight cryptography, homomorphic encryption, and post-quantum cryptography (PQC) have been proposed to enhance the security of IoT systems while accommodating their constraints.

  - Lightweight Cryptography: lightweight encryption plays a crucial role in securing smart device networks, offering both efficiency and a minimal resource footprint. Designed specifically for resource-constrained *IoT* devices, ightweight

encryption addresses the unique challenges posed by IoT environments. To meet the stringent requirements of IoT systems, encryption algorithms must be optimized to minimize space, memory usage, power demand, and energy consumption, ensuring they operate effectively without compromising device performance or battery life. This tailored approach makes lightweight encryption an ideal solution for safeguarding IoT ecosystems [15].

- Homomorphic Encryption (HE): HE is an advanced encryption method that enables computations to be executed directly on encrypted data without requiring decryption. This unique property ensures that sensitive data remains secure throughout the computational process.

  HE is typically based on asymmetric encryption (also known as public key encryption) and is particularly well-suited for applications requiring high levels of data privacy. It is widely used to protect sensitive information in smart grids, where encrypted energy usage data can be analyzed without exposing customer details, and in healthcare IoT applications, enabling secure processing of medical data while preserving patient confidentiality [16].

- **Machine Learning:** machine learning can play a crucial role in enhancing the security of IoT systems. By utilizing machine learning algorithms, IoT devices can detect anomalies in network traffic, identify potential threats, and predict security breaches before they occur. For instance, machine learning models can be trained to recognize normal patterns of behavior in IoT devices and flag deviations as potential security risks. Additionally, these models can continuously learn from new data, improving their detection capabilities over time. This proactive approach helps in preventing unauthorized access, attacks, and other security threats, making IoT environments more resilient and secure.

  - Supervised learning algorithms, such as Support Vector Machines (SVM), Decision Trees (DT), and Naive Bayes (NB), are commonly applied to secure IoT systems. However, these algorithms require substantial storage and significant time for data training. In contrast, unsupervised learning algorithms, such as K-means clustering and hierarchical clustering, do not require data training. While these unsupervised approaches are simpler to implement, they are generally less efficient compared to supervised techniques [7].
  - To overcome the limitations of traditional Machine Learning (ML) methods, deep learning techniques have been increasingly employed. Advanced deep learning algorithms, including Convolutional Neural Networks (CNNs), Re-

current Neural Networks (RNNs), Deep Belief Networks (DBNs), and Deep Q-Networks (DQNs), offer enhanced capabilities for improving the security of IoT systems, addressing complex threats more effectively [17, 18].

- **Blockchain:** blockchain is characterized as a fully distributed ledger, consisting of blocks that each contain a specific number of transactions and cryptographic links to the preceding block. Blockchain users rely on asymmetric key cryptography to securely log transactions [19]. As stated in [20], its appeal in a variety of fields stems from a few essential components:
  - Decentralized: it doesn't rely on central servers and runs entirely distributed.
  - Immutable: it is forbidden to remove any data from the blockchain. A new transaction containing the reverted data must be added to a new block in order to reverse an existing one.
  - Privacy: it preserves privacy by not requiring any identifying information and by having built-in mechanisms to ensure trust between nodes. It is important to stress that every node in the network can see the recorded data.
- **Intrusion Detection System:** an Intrusion Detection System (IDS) is a powerful tool designed to collect, monitor, and analyze user activities, networks, and services to detect and protect against intrusions that may compromise the confidentiality, integrity, and availability of an information system.The operations of an IDS can be divided into three key stages [21]:
  - Monitoring stage: uses network-sors to gather data.
  - Analysis stage: employs methods such as feature extraction or pattern identification to analyze the collected data.
  - Detection stage: relies on specialized techniques to identify and respond to potential intrusions.

  There are two main types of IDS:
  - Host-Based IDS (HIDS): HIDS is deployed on a single system to protect it from intrusions or malicious attacks that could compromise its operating system or data.
  - Network-Based IDS (NIDS): NIDS monitors network traffic to detect intrusions and malicious activities. It is primarily used to prevent attacks targeting the network. By analyzing network activity and node behavior, NIDS aims to identify intruders attempting to disrupt network operations.

- **Graph parameters-based solutions:** protecting applications from malicious behavior is one of the primary concerns in network application design. Therefore, the design must take into account the network's vulnerabilities that attackers might exploit.

  Typically, network application design relies on certain graph parameters, which involve selecting a set of nodes referred to as the *core*, such as a *maximal independent set* or a *dominating set*. Consequently, the security of the application depends on the security of this core. The principle is straightforward: the more *critical nodes* there are in the *core*, the more vulnerable the application becomes. Conversely, the fewer *critical nodes* there are, the more robust the application will be. In this context, the critical node detection problem serves as a valuable parameter to consider when designing secure network applications [14, 22].

There exist other advanced solutions for securing *IoT networks*, including fog computing-based approaches, edge computing-based approaches, and software-defined networking (SDN)-based approaches, among others. In this work, however, our focus is specifically on solutions leveraging graph parameter-based solutions to address the security challenges in *IoT networks*.

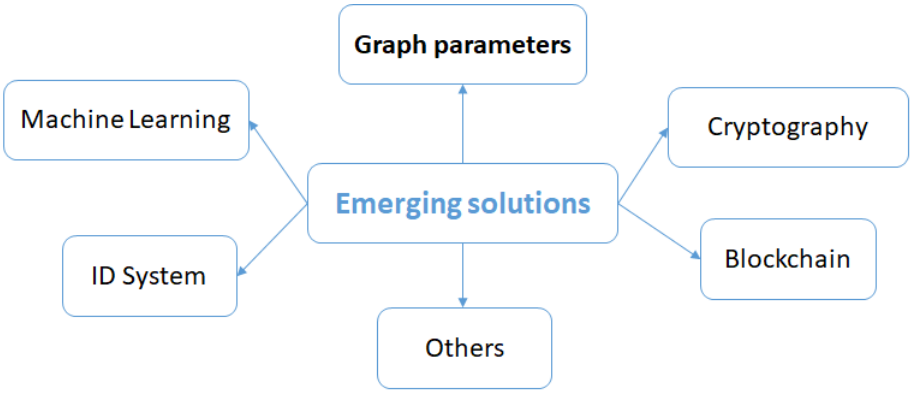


Figure 1.4: Emerging solutions for *IoT* security.

1.4 Graph parameters for network protection

The objective of this section is to present key graph parameters essential for *network protection*. It begins with a formal definition of the concept of a graph and its associated parameters. Subsequently, a detailed examination of the most well-known graph parameters utilized in *network protection* is provided. These include the *independent set, dominating set, and critical node set*, each of which plays a pivotal role in ensuring *network security* and resilience.

1.4.1 Graph and parameters

A graph is a mathematical abstraction that serves as a powerful tool for addressing a variety of problems. At its core, a graph $G = (V, E)$ consists of a finite non-empty set V of *nodes (vertices)* and a set $(E \subset V * V)$ of *links* that connect these nodes. The cardinality of the nodes set$(|V| = n)$ is called the *order* of G and the cardinality of the *edges* set $(|E| = m)$ is called the *size* of G.

Graph parameters are very useful in graph characterization and problems resolution. Generally, a real-life problem corresponds to a parameter in the graph modeling such one. A graph parameter is an invariant that depends only on the abstract structure of the graph (e.g., number of nodes or edges) and not on the graph representation (e.g., graph drawing). Determining a graph parameter leads clustering or decomposition of the graph nodes which facilitates the graph analysis. For example, finding the dominating set allows clustering the nodes set into dominants and dominated ones. Accordingly, the dominants can be used to transmit to all the other nodes in a communication network. Critical nodes detection and deletion allows decomposing graph into small components. As such, the problem complexity can be considerably by processing each component separately. Graph parameters are very useful in problem resolution but their determination is not always an easy task, and their related optimization problems are usually hard to solve.

To enhance clarity, before exploring the key parameters for network protection, we will first introduce the complexity classes of problems and their associated parameters. In general, solving a problem requires identifying the most effective algorithm for the task. The two primary factors in determining an algorithm's effectiveness are its execution time, which reflects the time required to complete the algorithm, and its space complexity, which indicates the memory needed for the algorithm's operation [23]. A complexity class refers to a family of problems characterized by a similar type of computational difficulty. In other words, it groups problems of equivalent complexity. Various complexity classes have been defined to categorize problems based on their computational cost, with the most notable ones being as follows [23, 24]:

- **Deterministic polynomial class (P):** a problem is said to be of class P, if for which we know an algorithm solution of complexity upper bounded by a polynomial of order t whose base x is a constant computed in terms of problem inputs such as the order n or size m of the graph $G = (E, V)$. The complexity of such an algorithm is denoted by $O(x^t)$, and the best algorithms complexities of P problems are: $O(log\, x)$, $O(x)$ or $O(x^2)$. Example of polynomial-time solvable problem is the problem of connectivity in a graph.

- **Non-deterministic Polynomial class(NP):** the class NP contains the decision problems that can be decided using a non-deterministic machine in polynomial time. A decision problem is a problem that can be solved in terms of yes/no. The NP problems admit a polynomial algorithm that is capable of testing the validity of a solution to the problem. Problems in this class can be solved by listing the set of possible solutions and testing them using a polynomial algorithm.
- **NP-Complete class:** NP-Complete problems signify the most computationally demanding issues within NP. If an algorithm were found capable of solving any NP-Complete problem in polynomial time, it would yield an efficient solution for all NP-Complete problems. At present, all recognized algorithms for NP-Complete problems necessitate time that increases superpolynomially relative to the input size.
- **NP-Hard class**. a problem is known to be NP-hard if it is harder than a NP-Complete problem, and there is a NP-complete problem that reduces to this problem.

The relationship between the classes of problems is illustrated in Figure 1.5. It is evident that a problem belongs to the NP-Complete class if it is a member of NP and is at least as difficult to solve as any other problem within NP.

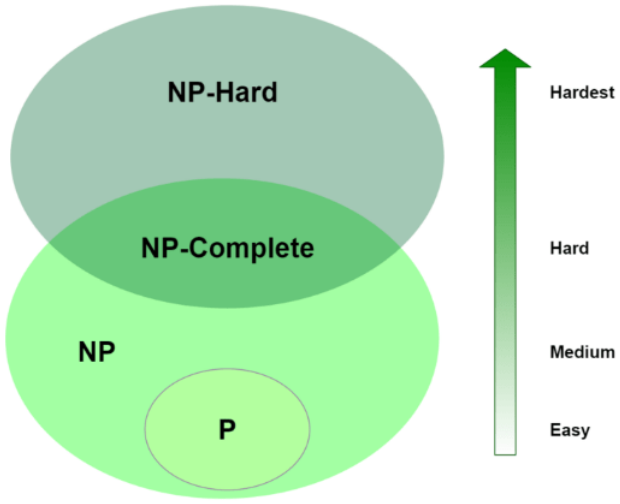


Figure 1.5: Relationship between classes of complexity.

To define the suitable approach for solving a given problem, we have first to set the problem complexity. Generally, the optimization problems can be solved with an exact algorithm or an approximate algorithm. In exact algorithms, the solutions are optimal

and their optimality is guaranteed. In case of NP-complete or NP-hard problems, exact algorithms run in non-polynomial time, which means it would take both important time and space to find a solution. However, approximate algorithms generate high quality solutions in a reasonable time, but there is no guarantee of finding an optimal solution [22, 25].

1.4.2 Dominating set

In a graph $G = (V, E)$, $D \subset V$ is a dominating set of G if every node is either in D or adjacent to some nodes in D. The minimum cardinality $\gamma(G)$ of a dominating set is called the domination number of G with its finding (minimum dominating set problem) is an NP-Hard problem [26]. The minimum dominating set problem has several applications mainly, in communication network to select the minimum set of computers to transmit the information to the remaining ones.

The literature identifies several variants of dominating sets, including the total dominating set, k-dominating set, connected dominating set, and weakly connected dominating set [27].

- **Total Dominating Set:** a set D is called a total dominating set (TDS) if every node in the graph has at least one neighbor in D, i.e., $\forall v \in V : N(v) \cap D \neq \emptyset$
- **k-Dominating Set:** a set D is referred to as a k-dominating set (KDS) if every node outside D has at least k neighbors within D.
- **Connected Dominating Set:** a connected dominating set (CDS) is a dominating set D where the subgraph induced by D is connected.
- **Secure Dominating Set:** a dominating set D of a graph G is said to be a secure dominating set (SDS) if every node $u \in V \setminus D$ is adjacent to a node $v \in D$ such that $(D \setminus \{v\}) \cup \{u\}$ is a dominating set of G.

In this thesis work, a lot of attention is given to this parameter by studying the secure domination, a very interesting variant in the graph security field (see Chapter 2).

1.4.3 Independent set

An independent set in a graph $G = (V, E)$ is a set of nodes with no two of which are adjacent. A maximum independent set is an independent set such that no other independent sets of G are larger. The maximum size $\alpha(G)$ of an independent set is called the independent number of G. Maximum independent set has many applications in different

fields, including information retrieval, signal transmission analysis and computer vision [22]. It is well known that the computation of the maximum independent set is NP-hard problem [28].

Given its significance and the variety of applications, *maximal independent set* has been extensively studied in the literature (see Chapter 3), leading to the development of numerous variants. Some well-known variants include the maximal weighted independent set [29–32], maximal independent set listing [33–37], 1-maximal independent set [38, 39].

- **Maximal Weighted Independent Set:** this variant extends the concept of a maximal independent set by associating weights with the vertices in the graph. The goal is to find a maximal independent set where the sum of the weights of the vertices is maximized. This variant is particularly useful in optimization problems where certain elements (represented by vertices) have different levels of importance or value.

- **Maximal Independent Set Listing:** in this variant, the objective is to enumerate all possible maximal independent sets in a given graph. While finding a single maximal independent set is computationally straightforward, listing all maximal independent sets is significantly more challenging due to their potential exponential number in large graphs. This variant is valuable in scenarios where a comprehensive analysis of all possible configurations is needed, such as in combinatorial optimization or network analysis.

- **1-Maximal Independent Set:** a 1-maximal independent set is a specific relaxation of the maximal independent set, where the independent set is required to be *maximal* only concerning vertices that are at most one edge away (i.e., neighbors) in the graph. This means that no single vertex adjacent to the set can be added without violating independence. This variant is often used in local optimization or heuristic algorithms where full global maximality is not required.

1.4.4 Critical node set

in graphs or networks, the significance of nodes varies, with some nodes playing a more critical role than others. A node is considered important if its failure or malicious behavior substantially impacts network performance or functionality. Depending on their role within the network, these nodes are referred to by various terms, such as most influential nodes, most vital nodes, most k-mediator nodes, or key-player nodes. When the importance of a node is specifically tied to network connectivity, these nodes are often categorized as Critical Nodes [22]. By definition, critical nodes represent the set of

nodes, deletion of which allows maximally destroyed the graph connectivity. The related optimisation problem is called critical nodes problem which seeks of determining the minimum set of nodes, deletion of which minimizing the network connectivity. According to the used metric for connectivity assessing, critical nodes problem has many variants mainly, maximizing the number of connected components, minimizing component size, component-cardinality-constrained critical node, etc [24].

- **Maximize the Number of Connected Components(MaxNum):** in this variant, the objective is to maximize the number of connected components in the graph. Specifically, the goal is to remove a predetermined number of nodes from the graph to disconnect its nodes as much as possible, thereby increasing the number of connected components.
- **Minimize the largest connected Component size (MinMaxC):** in this variant, the metric considered is the minimization of the size of connected components. *MiniMaxC* aims to find the set of k nodes whose removal minimizes the size of the largest connected component in the induced graph.
- **Component-Cardinality-Constrained Critical Node Problem (3C-CNP):** this variant aims to identify the minimal subset of nodes in a graph, whose removal produces connected components, each with a cardinality not exceeding a specified integer bound.

In this thesis, significant focus is placed on this parameter through the study of the 3C-CNP variant, which is a particularly intriguing topic in the field of network security (see Chapter 4).

1.5 Distributed Computing

Designing distributed computing systems is a multifaceted process that demands a deep understanding of the challenges involved, as well as the theoretical foundations and practical strategies for addressing them. This section begins by introducing the concept of distributed systems, examining communication models, and defining distributed algorithms. Subsequently, we explore various distributed computing platforms, highlighting their essential characteristics.

1.5.1 Distributed systems and Distributed algorithms

A distributed system is a collection of independent entities that collaborate to address a problem that cannot be resolved individually [40]. This concept is commonly applied

to communication networks and multi-processor computing systems. In such systems, each processor is capable of communicating only with its directly connected processors, referred to as neighbors.

A *distributed algorithm* is an algorithm designed to execute across the nodes of a distributed system. These algorithms are integral to various domains, including network communications, distributed information processing, and distributed computing. By leveraging only local knowledge, individual nodes can operate concurrently and communicate with one another to collaboratively solve a shared problem.

The fundamental communication models in distributed systems are the message-passing model and the shared-memory model [40, 41]:

- **Message-Passing model:** in the message-passing model, nodes within a distributed system communicate exclusively through the exchange of messages. In synchronous message-passing systems, communication occurs in distinct rounds: messages sent in round k are delivered to all recipients before any messages from round $k+1$ can be transmitted. Conversely, in asynchronous message-passing systems, messages are assumed to eventually reach their destinations, but the delivery times are unpredictable. Analyzing algorithms in asynchronous settings is more complex than in synchronous ones due to the inherent uncertainties and lack of timing guarantees.

- **Shared-Memory model:** in the shared-memory model, processes communicate by reading from and writing to a shared memory space. Synchronization plays a critical role in ensuring consistency and coordination among processes in such systems. Distributed shared-memory systems extend this model by implementing a shared-memory abstraction over a message-passing architecture, enabling the convenient use of shared-memory software modules in environments where physical shared memory is not directly available.

Usually, a distributed system can be effectively modeled as a graph $G(V, E)$, where V represents the set of vertices and E represents the set of edges. In this representation, the vertices correspond to the computing nodes of the distributed system, while an edge exists between two vertices if a communication link connects the corresponding nodes. Moreover, distributed systems can be characterized by two main properties: processor *anonymity* and algorithm *uniformity*.

- **System anonymity:** distributed systems are classified into two types based on processor anonymity:

  - Anonymous systems: in such systems, all processors are identical and indistinguishable from others with the same degree of connectivity in a deterministic manner. Each processor identifies its communication links using local numbers, referred to as port numbers.
  - Non-Anonymous systems: here, processors can be uniquely distinguished, typically through the use of unique identifiers.

- **Algorithm uniformity:** algorithm uniformity pertains to whether all processors execute the same algorithm:
  - Uniform algorithms: all processors execute an identical algorithm.
  - Non-Uniform algorithms: at least one processor executes a different algorithm.

1.5.2 Performance metrics

The performance of a distributed algorithm can be evaluated using various complexity measures, including time complexity, bit complexity, space complexity, and message complexity, as detailed below [41, 42].

- **Time complexity:**
  - For a synchronous distributed algorithm, Time refers to the number of communication rounds required for the algorithm to complete in the worst-case scenario. Each round typically involves all nodes performing computations and exchanging messages simultaneously.
  - For an asynchronous distributed algorithm, Time is measured as the number of discrete steps required for the algorithm to complete in the worst-case scenario. A step is typically defined as a single atomic action, such as a message being sent or a node performing a computation.

  For example, in a network represented by a graph $G(V, E)$, where V is the set of nodes and E is the set of edges, the time required to send a message to the farthest node may take up to $n-1$ steps in the worst case, where n is the number of nodes in the graph.

- **Space complexity:** the space complexity of a distributed algorithm specifies the maximum amount of memory, measured in bits, required for local storage at each node. This metric is particularly important in scenarios where nodes need to store large data structures, such as routing tables in routing algorithms.

- **Bit complexity:** bit complexity primarily refers to the maximum message length that can be transmitted. Generally speaking, this won't be an issue unless the message is big or gets enlarged while it travels over the network. For instance, an application that visits the network's nodes in a specific order might need a special message that contains the node identifiers. A maximum of n node identifiers will be included in the message in this scenario. Assuming that a node identifier requires $log\, n$ bits, the bit complexity for this example algorithm will be $O(n\, log\, n)$.

- **Message complexity:** message complexity represents the aggregate quantity of messages transmitted throughout the execution of a distributed algorithm. It serves as a key indicator of the communication cost associated with the algorithm. In many distributed applications, the cost of message communication is often the dominant factor influencing the overall performance and efficiency of the algorithm.

1.5.3 Distributed graph algorithms

the distributed processing of graph data has numerous practical applications and has been extensively studied. A graph algorithm operates on graph structures to derive solutions to problems represented within them. In the literature, graph problems are typically categorized as sequential, parallel, or distributed, depending on how the algorithms are executed within a computing environment [43, 44]:

- **Sequential graph algorithms:** a sequential graph algorithm follows a single flow of execution, processing operations in a step-by-step manner. It takes an input, performs computations on it sequentially, and produces the corresponding output.

- **Parallel graph algorithms:** parallel graph algorithms leverage multiple processing units to execute computations concurrently, rather than sequentially. By dividing the graph data and distributing tasks across multiple processors, these algorithms aim to improve efficiency and reduce execution time. Coordination mechanisms, such as synchronization and communication between processors, are essential to ensure correctness and consistency.

- **Distributed graph algorithms:** distributed graph algorithms operate in a decentralized manner, where each computational node corresponds to a vertex in the graph. These algorithms are designed to solve problems inherent to the network they represent, leveraging local computation and communication between nodes to achieve a global solution.

Designing parallel or distributed graph algorithms is often challenging and requires sophisticated techniques. The execution of operations may heavily depend on prior computations, necessitating extensive communication and synchronization among workers to ensure correctness and efficiency. Sometimes, the solution requires the algorithm to be both parallel and distributed. Table 1.1 highlights the key differences between parallel and distributed graph algorithms.

Table 1.1: Comparison of Parallel and Distributed Graph Algorithms

Aspect	**Parallel Graph Algorithms**	**Distributed Graph Algorithms**
Execution Model	Uses multiple processors that share memory or have fast interconnections.	Uses a distributed system where each node operates independently with local memory.
Communication	Processors communicate through shared memory or high-speed interconnects.	Nodes communicate over a network, often with higher latency.
Data Distribution	The graph is partitioned among processors but usually resides in shared or tightly coupled memory.	Each node typically holds only a portion of the graph and communicates with others over a network.
Synchronization	Requires synchronization mechanisms like barriers or locks to coordinate parallel execution.	Relies on message passing or asynchronous communication for coordination.
Scalability	Limited by memory bandwidth and inter-processor communication overhead.	More scalable due to distributed processing, but network latency can be a bottleneck.

In this thesis, significant attention is given to distributed graph algorithms by studying *secure domination, the maximal independent set, and critical node* problems. Before presenting an example of a distributed graph leader election algorithm, as described in [45], we first introduce essential message primitives and functions used in the algorithms discussed throughout this thesis (see Table 1.2).

Table 1.2: Functions of the proposed algorithms.

Function	Definition
$send(m, *)$	Broadcasts the message m
$send(m, id)$	Transmits the message m to the node identified by id
$read()$	Waits for the receipt of messages in a blocking manner. If no message is received, the function remains blocked at this instruction until a message is received
$read(wt)$	Awaits the receipt of messages. If no message is received after wt milliseconds, execution will proceed to the subsequent instruction.
$getId()$	Returns the node identifier
$getNNeig()$	Get the total number of adjacent nodes of the specified node.
$stop()$	Halts the execution of the program

In this example, the leader is defined as the node with the minimum value among all nodes in the graph. To achieve leader election, each node waits for a duration proportional to its value before transmitting its message. Consequently, the node with the smallest value initiates the first message transmission. This message propagates through the network using a flooding process, for instance, to inform all other nodes. The pseudo-code for this example algorithm is presented in Algorithm 1.1.

Algorithm 1.1: Distributed graph algorithm example

```
Data: x, gt
Result: Leader
begin
    Leader ← false;
    once ← false;
    t ← x * gt;
    while (true) do
        M ← read();
        if (M = Null) then
            Leader ← true;
            send(A, *);
            stop();
        if (M = A and once = false) then
            oncer ← true;
            send(A, *);
            stop();
```

1.5.4 Distributed computing platforms

Distributed computing platforms and paradigms have undergone remarkable evolution over the past few decades, transitioning from large, room-sized resources (processors and memory) to compact, highly efficient computing nodes. Today, distributed computing capabilities are integral to nearly all modern application domains, enabling advanced functionalities and seamless scalability. Among the most prominent distributed computing platforms and paradigms, as illustrated in Figure 1.6 and discussed in [41, 46], are:

- **Grid computing:** a grid consists of loosely coupled, heterogeneous, and geographically dispersed computing elements, interconnected by a network to collaboratively execute large-scale tasks. These computationally intensive scientific tasks may encompass a wide range of applications, including seismic analysis, drug discovery, and bioinformatics challenges. Grid computing enhances the utilization of underutilized processing power, thereby reducing task completion time through parallelization

  Grid networks are overseen by a central control node, typically implemented as a server or a set of servers. This central node is responsible for managing and coordinating the resources and computational tasks within the network pool. A grid node contributes resources—such as memory, processing power, and storage—to the network pool. These nodes actively engage in the execution of computations within the distributed grid [47].

- **Cloud computing:**
  cloud computing emerged from grid computing with the aim of delivering computing resources as a service to users, thereby expanding the object-oriented programming paradigm. It provides computation, software applications, data access, data management, and storage without necessitating users' awareness of the location or specifics of the underlying infrastructure. Grid computing may be integrated into the cloud, contingent upon the application and user specifications. Cloud and grid computing emphasize scalability, utilizing load balancing to attain this objective. In grid computing, a singular task is divided into smaller tasks executed concurrently across multiple processors to enhance the utilization of available computational resources. Conversely, cloud computing encompasses more than mere processing power, providing services including website hosting, database support, and additional functionalities. In general, cloud computing offers a more extensive array of services compared to grid computing.

  The Cloud providers offer services in various standardized formats, as noted in [48]:

  - Infrastructure-as-a-Service (IaaS): it involves the provision of hardware to the client, who assumes responsibility for the installation of all stack components on the hardware, which includes operating systems, middleware, runtime environments, and applications. The cloud provider is exclusively accountable for the management of the hardware component.
  - Platform-as-a-Service(PaaS): the supplier's responsibility is elevated further up the stack, as they will be responsible for installing and managing operating systems, middlewares, and all required execution environments.
  - Software-as-a-Service (SaaS): it involves client interaction with cloud services via a Graphical User Interface. All essential services are provided as ready-to-use applications, with the provider responsible for the complete infrastructure stack.

- **Wireless Sensor Network:** a wireless sensor network (WSN) comprises numerous small computing nodes, each equipped with sensing and wireless communication capabilities. These networks gather environmental data and transmit it to a central node for further analysis using multi-hop communication. WSNs have a broad range of applications, including habitat monitoring, military surveillance, and target tracking. As large-scale distributed systems, WSNs rely on scalable distributed algorithms to address challenges such as data aggregation, topology control, and routing.

  The end-nodes in WSNs transmit their collected data to a central unit, referred to as the sink or gateway node. The gateway serves either as a bridge between two or more distinct networks or as an interface between the network and its users. While the sink or gateway node is typically static, it can also be mobile in certain applications [49].

- **Fog computing:** fog computing seeks to enhance cloud functionalities by relocating computational resources nearer to edge networks. A system of geographically distributed computing devices is defined as one that connects various heterogeneous devices at the network's edge, independent of exclusive reliance on seamless integration with cloud services. Fog computing functions as a vital intermediary between the network edge and the cloud. Devices situated at the fog's edge enable the implementation of new IoT applications that demand low latency and high performance standards [50, 51].

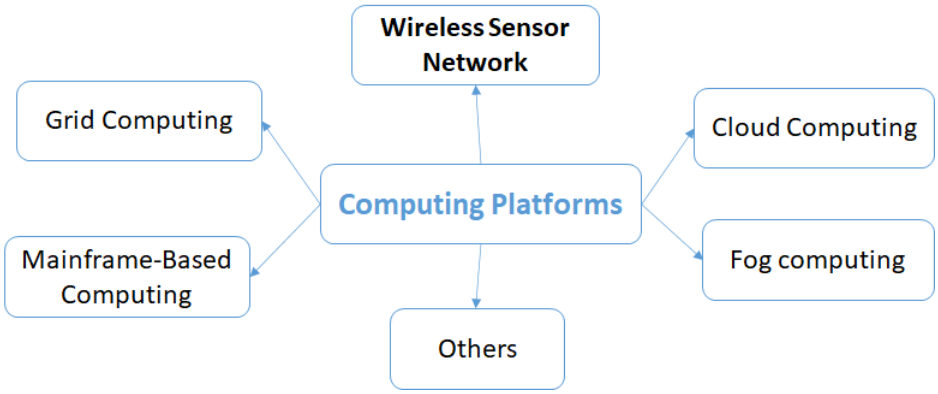


Figure 1.6: Distributed computing platforms.

1.6 Conclusion

In this chapter, we presented and discussed the basic concepts of *IoT*, including its architecture, applications, and security challenges, along with emerging solutions. Additionally, the chapter elaborated on graph parameters and their importance in *network protection*, introducing key parameters such as the *domination set, maximal independent set, and critical node set*. Finally, it highlighted the role of distributed computing, examining its systems, algorithms, and platforms, which collectively establish the theoretical foundation for the subsequent chapters of this thesis.

In the next chapter, we will focus on *the secure dominating set parameter* as a key aspect of *IoT network protection*, which constitutes the first contribution of this thesis. Additionally, we will present distributed solutions designed to effectively address this critical challenge.

Chapter 2

Distributed Detection of Secure Dominating Sets in IoT Networks

2.1 Introduction

Dominating nodes in networks play a crucial role as monitoring points for detecting malicious activities, ensuring comprehensive oversight of devices. Additionally, they act as secure aggregators, collecting and verifying data before transmitting it to central servers, thereby reducing communication overhead and minimizing vulnerability to potential attacks. The challenge of identifying these nodes is known as the *Dominating Set Problem (DSP)*. Based on the properties of the resulting set, several variants have been proposed, with one of the most significant being the *Secure Dominating Set Problem (SDSP)*.

In this chapter, we introduce a heuristic-based distributed method to address the *SDSP* in *IoT networks*. The *SDSP* is a variant of the *DSP* that incorporates security and resilience constraints. We begin by introducing the problem statement, highlighting the specific challenges, and providing a comprehensive literature review related to the *SDSP*. Next, we provide the theoretical foundations necessary to effectively address the problem. To frame the context of the current work, we present and examine prior results on *SDSP*, thereby establishing a foundation for our contributions. Finally, we present the design and analysis of our proposed solution, including a detailed description of the proposed algorithm and an evaluation of its computational complexity.

2.2 Problem statement and definitions

We consider an *IoT network* as a simple undirected connected graph $G = (V, E)$ where V is a set of n devices or processors (nodes) and E represents the set of m bidirectional

communication links (edges). We denote by $d(v)$ the degree of any node v. And Δ represents the maximum node degree in the graph G.

2.2.1 Theoretical definitions

In the following, we provide the basic theoretical definitions necessary for addressing the *SDSP*.

Definition 2.1 (Dominating Set)**.** A dominating set X of G is a subset of V such that every node in V either belongs to X or has a neighbor in X.

Definition 2.2 (Minimal Dominating Set)**.** A dominating set (DS) is a minimal dominating set (MDS) if no proper subset of MDS is a dominating set.

Definition 2.3 (Minimum Dominating Set)**.** A minimum dominating set (MinDS) of a graph G is a dominating set X with the smallest possible number of nodes among all dominating sets of G.

Definition 2.4 (Domination Number)**.** The minimum cardinality $\gamma(G)$ of a dominating set is called the domination number of G.

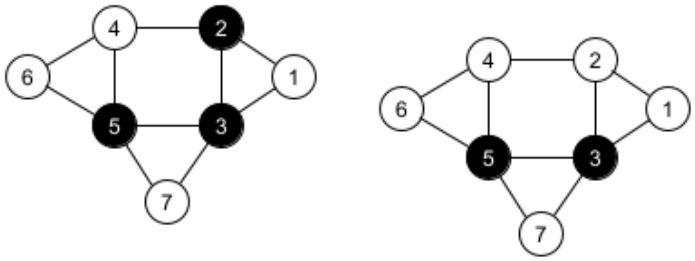
Definition 2.5 (Secure Dominating Set)**.** A dominating set X of a graph G is said to be a secure dominating set (SDS) if every node $u \in V \setminus X$ is adjacent to at least one node $v \in X$ such that $(X \setminus \{v\}) \cup \{u\}$ is a dominating set of G.

Definition 2.6 (Minimal Secure Dominating Set)**.** A secure dominating set (SDS) is a minimal secure dominating set (MSDS) if no proper subset of MSDS is a dominating set.

Definition 2.7 (Minimum Secure Dominating Set)**.** A minimum secure dominating set of a graph G is a secure dominating set X with the smallest possible number of vertices among all secure dominating sets of G.

Definition 2.8 (Secure Domination Number)**.** The secure domination number, denoted by $\gamma_s(G)$, is the minimum cardinality of an SDS of G.

To illustrate these theoretical definitions, consider a graph G with seven nodes labeled 1, 2, 3, 4, 5, 6, and 7. In Figure 2.1a, nodes 2, 3, and 5 form a minimal secure dominating set. In contrast, in Figure 2.1b, nodes 3 and 5 form a minimal dominating set but not a minimal secure dominating set, because there is an issue with node 7. If we remove node 5 and replace it with node 7, node 6 is not dominated. Similarly, if we remove node 3 and replace it with node 7, node 1 is not dominated.



(a) a minimal secure dominating set

(b) a minimal dominating set

Figure 2.1: Illustration of the theoretical definitions with a graph G of $\gamma(G) = 2$ and $\gamma_s(G) = 3$.

2.2.2 Problem definition

The *Secure Dominating Set Problem (SDSP)* is a computational combinatorial problem that seeks to find the smallest possible secure dominating set for a given graph G. It involves determining a set $X \subset V$ that satisfies both the dominating and security (each node $u \notin X$, there must be a neighboring node v of u within X such that the swap set $(X \setminus v) \cup u$ remains a dominating set) conditions while minimizing the cardinality of X.

Secure Dominating Set Problem
Input : A graph $G = (V, E)$
Output : A Secure dominating Set of nodes ($X \subset V$)

In the literature, it is well known that the computation of the domination and secure domination numbers is NP-hard [26, 52]. This implies that no known efficient algorithms exist to solve these problems for all instances in polynomial time.

2.2.3 Literature review

Historically, the concept of *secure domination* has been introduced in 2005 by Cockayne et al. [53], and has been investigated in a series of papers [54–59]. In [53] several bounds and exact values for some classes of graphs have been established. Mynhardt et al. [58] Characterized trees with equal domination and secure domination numbers and introduced the concept of excellent trees. In [60], the authors characterized trees with equal independent domination and secure domination numbers. The relations between the *secure domination* number and other parameters such as domination, total domination, and independence numbers have been studied [52, 57]. Several generalizations of *secure*

domination have been introduced. Burger et al. [61] extended the concept of *secure domination* by introducing infinite secure domination, which concerns the perpetual security of a graph (protection of the graph against an infinite number of consecutive attacks). Goddard et al. [62] introduced eternal m-security by allowing several moves simultaneously to an attack which allows for decreasing the infinite secure domination number. We notice that a few proposed resolution algorithms are found in the literature based on a centralized approach.

The *SDSP* is NP-hard even when restricted to bipartite graphs and split graphs [52]. Due to their difficulty, only a few methods have been explored in the literature to solve them. Considering particular classes of graphs, the *SDSP* has been studied on trees and proper interval graphs. An algorithmic study has been carried out by Burger et al. [63] who proposed a linear time algorithm for finding a minimum *secure dominating set* and therefore the value of the *secure domination* number for a tree topology, while Araki and Miyazaki [64] provided a linear time algorithm for finding the secure domination number in proper interval graphs. To solve the *SDSP* on general graphs, two works were found in the literature. In the work of [65], the authors developed two exponential-time algorithms to address the *SDSP*. The first algorithm utilizes a branch-and-reduce method, while the second employs a classical branch-and-bound method. In the work of [66], the authors applied algorithmic game theory to study the *SDSP* in a multi-agent system. They proved that every Nash equilibrium is a minimal *secure dominating set* and a Pareto-optimal solution. Burger et al. [65] presented a simple heuristic that can be used to find an upper bound on the secure domination number of a connected graph. They showed that a *secure dominating set* X of a graph G can be obtained by computing a spanning tree of G and then determining the *secure dominating set* from the spanning tree.

On the other hand, with the rapid progress of wireless network technology, *IoT* networks have become increasingly popular for network monitoring, requiring minimal human intervention. The *DSP* is particularly useful in several contexts, including network coverage, data aggregation, routing, and security [67]. In the realm of *IoT* networks, this problem has been extensively studied in the literature [67–71]. The dominating nodes are equipped with enhanced security features, such as stronger encryption, tamper detection, or intrusion detection systems (IDS). However, in a standard *DSP*, each sensor is only responsible for detecting security issues within the system and lacks the capability to address them. To overcome this limitation, the *SDSP* can be employed [66]. By securing the dominating set, we ensure that even if some non-dominating nodes are compromised, communication remains secure because the critical paths in the network are protected.

The objective of this thesis chapter is to investigate the *SDSP* within the context of *IoT* networks. To address this, we propose a novel *distributed algorithm* aimed at

identifying a *secure dominating set* within these networks. Our methodology builds upon the heuristic approach introduced by Burger et al. [65], which we have mentioned earlier, and incorporates key aspects of their framework. Furthermore, as our solution leverages *spanning tree* construction, we draw upon several related works in this area to provide context and validation. For instance, in [72], the authors introduced a distributed algorithm for leader election. After finding a *spanning tree* of a wireless sensor network, each leaf routes a message through its branch to the root to find the leader in that branch. The root then elects the global leader from the received branch leaders. The message and time complexity of their algorithm is $O(n)$.

In [73], the authors proposed a leader election algorithm based on a tree routing mechanism for arbitrary networks. This algorithm begins with local leaders who initiate the flooding process to construct a *spanning tree*. During this process, each leader's value is routed through the network. When two *spanning trees* intersect, the tree with the better value continues the flooding process, while the other tree stops. By the end of the algorithm, only one *spanning tree* remains, and its root represents the leader. In [74], the authors introduced an efficient and fault-tolerant algorithm called DoTRo, based on a tree routing protocol. The algorithm begins with local leaders initiating a flooding process to establish a *spanning tree*. During this process, their values are propagated. When two *spanning trees* intersect, the tree with the superior value continues, while the other ceases its process. The surviving tree's root becomes the leader. The algorithm's total complexity is $kn(m+1)$ messages, where n is the network size, m is the number of local minima, and k represents the number of neighbors per node.

2.3 Context of the work

In this section, we present some key results that are fundamental to the development of our algorithm, along with an intriguing heuristic for the *SDSP*, identified in the literature.

2.3.1 Previous results

In this subsection, we introduce some essential theoretical results that play a central role in developing our *distributed algorithm* for the *SDSP* in *IoT* networks. These preliminary findings provide a basis for understanding the structural properties of certain graph components, such as *end-vertices, support* vertices, and tree paths, which influence secure domination in graph structures. By exploring these properties, we are able to build a foundation for our proposed algorithm and offer insights into existing bounds and techniques from prior literature. These results guide the algorithm's efficiency and accuracy

when determining *secure dominating sets* in complex network topologies.

An *end-vertex* in a graph is a vertex with a degree of 1, meaning it is connected to exactly one other vertex. A *support* vertex, on the other hand, is a vertex with a degree of at least 2 and is directly connected to an *end-vertex*. In the context of trees, an *end-vertex* is commonly referred to as a *leaf*, signifying that it is a terminal point with no further extensions. A vertex in a tree T that has a degree of 3 or more is called a *branch* vertex, as it acts as a junction where multiple paths diverge. If a tree T has no branch vertices, then T is a path. An *endpath* P of a tree T is a subpath of T that contains a *leaf* l of T and in which every vertex $v \neq l$ has degree 2 in T. An *end-cluster* in a tree is a specific type of connected subtree formed by a *support* vertex and the *leaves* that are directly connected to it, essentially representing a localized grouping of terminal points.

In [53], several general bounds were established for the secure domination number $\gamma_s(G)$, and specific values of $\gamma_s(G)$ were determined for various graph classes, including paths, cycles, complete multipartite graphs, and products of paths and cycles. In [53, 63] it was proven that:

Theorem 2.1. *([53]). For every graph G, $\gamma_s(G) \geqslant \gamma(G)$.*

Theorem 2.2. *([53]). If T is a path P_n of order n, then $\gamma_s(G) = 3n/7$.*

Theorem 2.3. *([63]). Let T be a tree and let P be an endpath of order j in T. Then there is no SDS of T containing fewer than $3(j-1)/7$ vertices of P.*

This previous result is central to the development of our algorithm. For more details, the reader can refer to [53].

2.3.2 Centralized heuristic for the SDSP

In this subsection, we present an interesting centralized heuristic for determining a *secure dominating set* in a general graph, as proposed by Burger et al. [65]. This heuristic is based on the principle that the leaves of a tree securely dominate its *end-clusters*. To construct a *secure dominating set* X for a graph G, the method first computes a spanning tree of G and includes all the leaves of the tree in X. The *end-clusters* are then pruned, reducing the tree's size. This pruning process is repeated for each new smaller tree, with newly formed leaves being added to X, until all vertices of the original *spanning tree* are pruned. This iterative process ensures an efficient determination of a *secure dominating set*.

In the following, we illustrate how the proposed centralized heuristic constructs a *secure dominating set* using the example provided in Figure 2.2. Specifically, we consider the topology of the graph G depicted in Figure 2.2a.

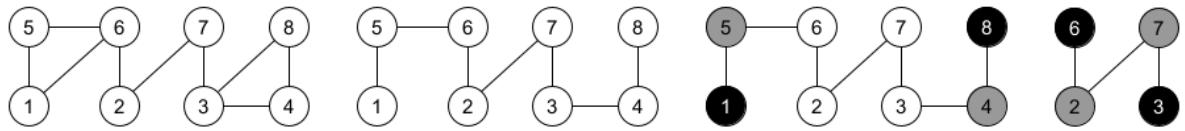


Figure 2.2: Example of executing the heuristic for a Secure Dominating Set on a graph with $\gamma_s(G) = 3$.

First, a *spanning tree* T of G is shown in Figure 2.2b. During the first iteration, the two leaves, 1 and 8 (indicated as solid vertices in Figure 2.2c), are inserted into the *secure dominating set* X. Next, the two *end-clusters* (the *support* vertices highlighted in grey) are removed from the tree, resulting in the smaller tree depicted in Figure 2.2d. The two leaves, 3 and 6, of this smaller tree are also inserted into X, after which the entire tree is pruned away. This execution results in the *secure dominating set* $X = \{1, 3, 6, 8\}$ with a cardinality of 4 for the graph G.

2.4 Algorithms design and analysis

In this section, we adopt the method proposed by Burger et al. [65], as described above, to identify a *secure dominating set* in a distributed manner. The method is structured into two distinct phases. The first phase involves the construction of the *spanning tree* for the *IoT* network, while the second phase focuses on deriving the *secure dominating set* from the constructed *spanning tree*. To provide a comprehensive introduction to these two phases, we will first define the variables and functions employed in our algorithm.

2.4.1 Variables and primitive functions

All the primary variables and fundamental functions used by the various algorithms are described in Tables 2.1 and 1.2, respectively.

2.4.2 Spanning tree construction

The initial phase involves constructing a *spanning tree* for the *IoT* network using the algorithm proposed by Bounceur et al. [72]. This algorithm employs the flooding protocol, which facilitates spanning tree construction by broadcasting each received message across all outgoing links, except the one from which it was received [75]. To prevent redundancy, any repeated messages received should be discarded. The pseudocode for the flooding

Table 2.1: Description of the variables

Variables	Description
d	Integer variable used to store the degree of a node
S	Integer variable used to store the number of messages received from current node neighbors.
$leaf$	Boolean variable. It takes the value true when it become a leaf node of a spanning sub tree
$support$	Boolean variable. It takes the value true when it become a support node of a spanning sub tree
SDS	Boolean variable that indicates whether or not the node in the secure dominating set

algorithm, as presented in Algorithm 2.1, illustrates this process in its simplest form, initiated by a designated node referred to as the root. The message complexity of Flooding algorithm is $O(m)$ where m is the number of edges of G, and the time complexity of Flooding is $O(dia)$ where dia is the diameter of G.

Algorithm 2.1: Flooding

```
Data: idRoot
begin
    id ← getId();
    visited ← false;
    if (id = idRoot) then
        send(A, ∗);
        visited ← true;
    while (true) do
        M ← read();
        if (M = A & visited = false) then
            send(A, ∗);
            visited ← true;
```

This process ensures the nodes communicate in a way that forms a *spanning tree* across the network. Once the *spanning tree* is established, a second algorithm is employed to identify its *leaf* nodes. To identify the leaves in the constructed spanning tree, we employ the Flooding for Leaf Finding (FLF) algorithm proposed by [72]. The pseudocode for this algorithm is provided in Algorithm 2.2. It builds upon the previously introduced Flooding Algorithm 2.1, utilizing two types of messages: $T1$ for the flooding process and $T2$ for acknowledgments.

The algorithm operates as follows: during the flooding process, after each $T1$ message is transmitted (lines 5–6 for the root node and lines 17–18 for other nodes), the transmitter awaits an acknowledgment in the form of a $T2$ message (lines 12 and 20). Initially, with the exception of the root node (line 3), every node is considered a leaf (line 8). However, if a transmitter receives any acknowledgment, it is reclassified as a non-leaf node (lines 20–22).

Algorithm 2.2: FLF

```
Data: idRout
Result: leaf
begin
    id ← getId();
    if (id = idRoot) then
        leaf ← false;
        once ← true;
        M ← (T1, id);
        send(M, ∗);
    else
        leaf ← true;
        once ← false;
    while (true) do
        (type, idR) ← read();
        if (type = T1 & once = false) then
            once ← true;
            M ← (T2, id);
            send(M, idR);
            M ← (T1, id);
            send(M, ∗);
        if (type = T2) then
            leaf ← false;
```

2.4.3 Secure dominating set algorithm for tree

This subsection introduces the second phase, providing a comprehensive description of the proposed algorithm, accompanied by a practical example to demonstrate its operation. The discussion concludes with a detailed complexity analysis, emphasizing the algorithm's efficiency in tree-based network architectures.

2.4.3.1 Description of SDSTree algorithm

In the following, we provide a comprehensive description of our proposed algorithm, called *SDSTree*, which will be used in the second phase. Each node in our algorithm maintains a set of local variables, as defined in Table 2.1, which contribute to achieving the global objective. The algorithm utilizes the *leaf* variable as a boolean input, with its value determined through the execution of the leaf-identifying algorithm.

Algorithm 2.3: SDSTree

```
Data: leaf
Result: SDS
begin
    S ← 1;
    d ← getNNeig();
    SDS ← false;
    support ← false;
    while (true) do
        if (leaf = true) then
            SDS ← true;
            send(0, ∗);
            Stop();
        else
            M ← read();
            if (M=0) then
                support ← true;
                send(1, ∗);
                Stop();
            else
                S ← S + M;
                if (S = d) then
                    leaf ← true;
```

Below is a detailed description of the *SDSTree* algorithm: Each node begins by setting its *SDS* and *support* state variables to false, initializing the value of variable S to 1, and obtaining its degree d using the *getNNeig()* function (see lines 2–5). If a node identifies itself as a *leaf* ($leaf = true$), it marks itself as part of the *SDS* by setting the *SDS* variable to true. The *leaf* node then broadcasts a message containing the value 0 to all its neighbors and stops execution (see lines 7–10). However, *non-leaf* nodes continuously

read messages from their neighbors. If a *non-leaf* node receives a message with value 0 from a *leaf* node, it becomes a *support* node, broadcasts a message containing the value 1 to signal its transition, and then stops execution (see lines 14–17). On the other hand, if a *non-leaf* node receives a message with value 1, it increments its local variable S by the value of the message received (see lines 12–17). If the sum S is equal to the node's degree d, indicating that it has received the necessary updates from all its neighbors and has become a *leaf* node, it sets the value of its $leaf$ variable to $true$ to signal its transition (see lines 19–20). The stopping condition of the *SDSTree* algorithm for each node occurs when they finalize their state, either as part of the *SDS* or by confirming their status as a *support* node.

2.4.3.2 Illustrated example

The objective of this example is to visually explain how the algorithm processes the IoT network, illustrating how *leaf* and *support* nodes communicate and how the *secure dominating set* is formed through iterative message exchanges between nodes. We consider an IoT network of arbitrary topology, composed of 12 nodes labeled 1 to 12, and represented in Figure 2.3a. The execution of the *spanning tree* algorithm on this network has resulted in the *spanning tree* shown in Figure 2.3b. For the *spanning tree* found, we observe that there are 4 *end-clusters*. The first *end-cluster* is composed of *support* node 2 and its *leaf* node 1; the second *end-cluster* consists of *support* node 3 and its *leaf* node 4; the third *end-cluster* comprises *support* node 9 and its *leaf* node 10; and the fourth *end-cluster* consists of *support* node 11 and its *leaf* node 12. As a result, the 4 *leaf* nodes 1, 4, 10, and 12 mark themselves as part of the *SDS* and begin sending messages with a value of 0 before stopping execution (Figure 2.3c). Subsequently, nodes 2, 3, 9, and 11 receive messages with a value of 0, sent by the previous *leaf* nodes, and become *support* nodes, and sending messages with a value of 1 before quitting execution (Figure 2.3d). Next, nodes 5 and 8 receive messages with a value of 1 and increment the variable S, which corresponds to the degrees of these nodes. Consequently, they become *leaf* nodes, mark themselves as part of the *SDS*, and send messages with a value of 0 before stopping their execution (Figure 2.3e). Finally, nodes 6 and 7 receive messages with a value of 0, sent by the *leaf* nodes, and become *support* nodes. They then broadcast messages with a value of 1 before completing the execution (Figure 2.3f). Ultimately, all nodes in the network complete their execution, with the marked nodes displayed as shown in (see Figure 2.3g).

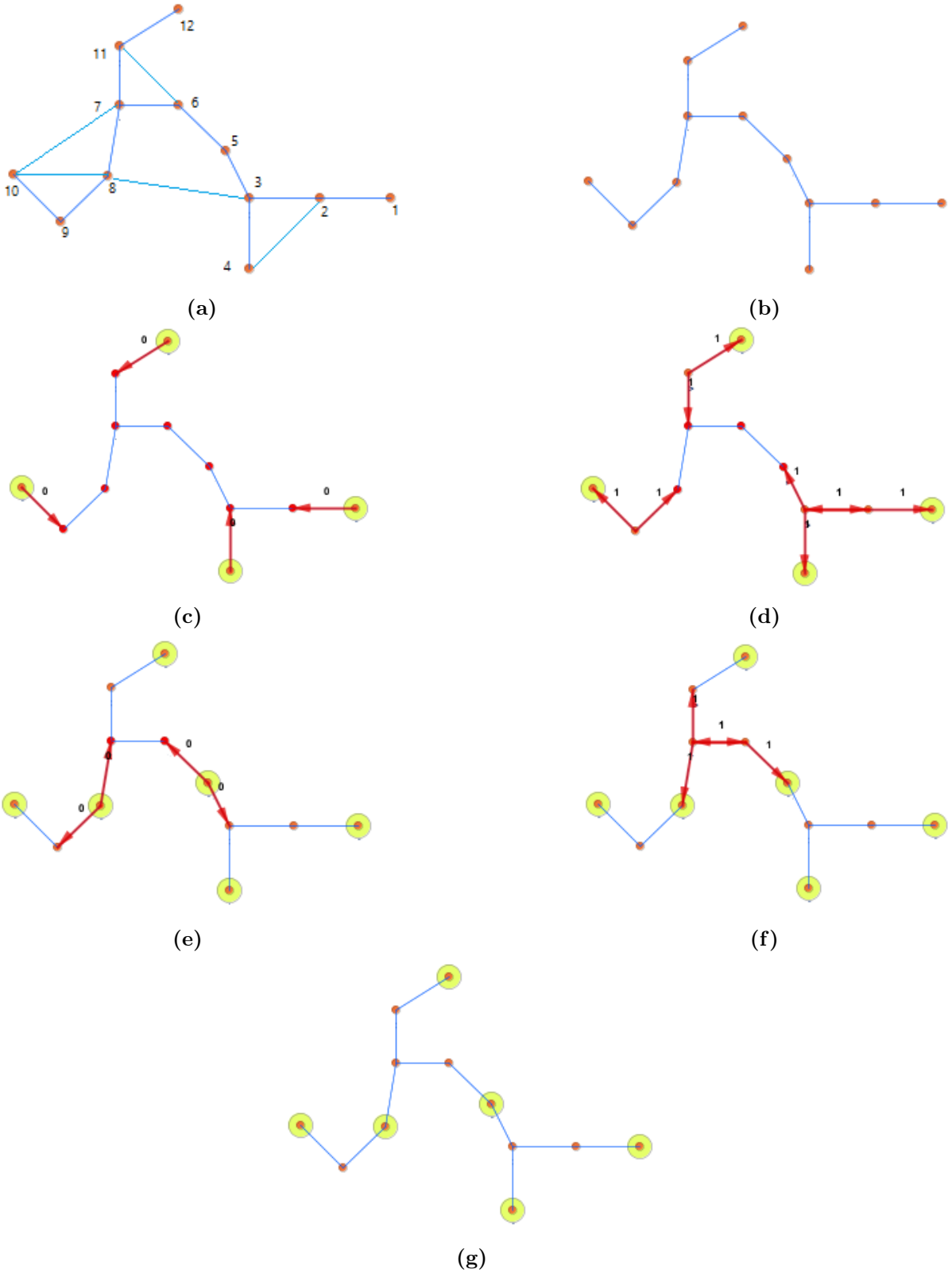


Figure 2.3: Illustration of the *SDSTree* algorithm.

2.4.3.3 Complexity

In this sub sub-section, we will analyze the complexity of the proposed *SDSTree* algorithm by evaluating the number of iterations required for its execution and the total number of messages sent.

Proposition 2.1. *The SDSTree algorithm finds a secure dominating set of nodes in a given IoT network after* $\Delta - 1$ *iterations.*

Proof. Consider an *IoT network* modeled as a connected undirected graph $G = (V, E)$ with n nodes and m edges. We will use the notations introduced earlier (see Section 2.2). Each node $v \in V$ has $d(v)$ neighbors, were $1 \leq d(v) \leq \Delta$. In the spanning tree of G, we distinguish two types of nodes: *leaf* nodes and *support* nodes. *Leaf* nodes are those with $d(v) = 1$, while *support* nodes are those with $d(v) > 1$.
The execution of the algorithm begins with the *leaf* nodes, which each enter the while-loop exactly once at the start. During this entry, the *leaf* nodes mark themselves as part of the *secure dominating set* and send messages with a value of 0 to their parent nodes, which are designated as *support* nodes. For a node to become a *support node*, it requires a single iteration to check if it has received a message with a value of 0 from any of its *leaf* neighbors. After this, the remaining *non-leaf* nodes, in subsequent phases of the heuristic, must check the sum of the message values sent by their neighboring *support* nodes. In the worst case, a *non-leaf* node needs to ensure that the sum of the messages is at most $d-1$, where d is the degree of the node. Since the maximum degree in the graph is Δ, the worst-case number of iterations needed for any node to complete its transformation into a *leaf* is $\Delta - 1$. Thus, the algorithm's overall worst-case number of iterations is $\Delta - 1$.
□

Proposition 2.2. *Given an IoT network, the SDSTree algorithm requires a total number of communication messages equals to* n *messages.*

Proof. Each *leaf* node sends exactly one message with a value of 0 to its corresponding *support* node. likewise, Each *support* node sends exactly one message containing information with a value of 1. Given that the network consists solely of these two types of nodes (*leaf* nodes and *support* nodes), it follows that each node in the network sends precisely one message. Since there are n nodes in total, the total number of messages sent in the network is the sum of messages sent. Thus, the total number of messages sent is equal to n.
□

2.5 Conclusion

In this chapter, we have proposed a distributed solution for identifying secure dominating nodes in arbitrary networks. To provide a clear understanding of our solution, we began by presenting the problem statement along with the essential theoretical definitions necessary for addressing the SDSP. We then contextualized our work by reviewing previous research, highlighting existing challenges, and identifying open problems in this field. Finally, we provided a comprehensive overview of the design and analysis of the proposed algorithm, detailing its structure and evaluating its computational complexity.

In the next chapter, we will shift our focus to the maximal independent set parameter, the second key contribution of this thesis, where we will propose distributed solutions to enhance its applicability.

Chapter 3

Distributed Construction of Maximal Independent Sets in IoT Networks

3.1 Introduction

The *Maximal Independent Set Problem (MISD)* is a powerful tool for enhancing the security, resilience, and efficiency of *IoT networks*. By strategically selecting non-adjacent nodes as key communication and security points, it helps mitigate attacks. These nodes can function as distributed intrusion detection points, preventing malicious nodes from overwhelming specific network regions. Additionally, *maximal independent set* can be used to identify *critical nodes*, further strengthening the network's stability and security.

In this chapter, we present a *distributed algorithm* for identifying maximal independent nodes in arbitrary *IoT networks*. We begin by introducing the theoretical definitions and formulating the problem statement for the *MISP*. To contextualize our work, we provide a comprehensive review of the state-of-the-art in *MISP*, categorizing and critically analyzing the most pertinent distributed algorithms proposed in existing literature. Furthermore, we outline the methodology employed in the design of our algorithm and discuss related research in the field. Subsequently, we introduce our innovative *distributed MISP algorithm*, providing a thorough description of its design, detailing its various stages, and performing a complexity analysis to assess its efficiency.

3.2 Problem definition

The *Maximal Independent Set Problem (MISP)* is a computational combinatorial problem that seeks to identify a large possible set of non-adjacent nodes *(MIS)* in the given graph or network, where the addition of any other node would compromise the independence of

the set. Formally, the *MISP* takes as input a graph $G = (V, E)$, and returns as output a set of nodes $MIS \subset V$.

Maximal Independent Set Problem
Input : A graph $G = (V, E)$
Output : A Maximal Independent Set of nodes ($MIS \subset V$)

In the literature, it is well known that the computation of independent and domination numbers is NP-hard [26, 28]. This implies that no known efficient algorithms exist to solve these problems for all instances in polynomial time.

3.3 Context of the work

This section provides the necessary background to understand the foundations of the proposed approach. It begins with an overview of previous research on distributed solutions to *MISP*. Following this, it presents an overview of existing distributed *Wait Before Starting* algorithms, which are relevant to the coordination mechanisms explored in this work.

3.3.1 Existing MISP algorithms

The concept of distributed computation for *MISP* was first introduced in the pioneering studies of Valiant [76] and Cooley [77] in the early 1980s. These foundational studies laid the foundation for subsequent research in the area, playing a critical role in the development of *distributed algorithms* for *MISP*. Since then, significant advancements have been achieved in the domain of distributed computing, including the design of more sophisticated algorithms and the exploration of various graph structures and network topologies. Numerous parallel and *distributed algorithms* have been rigorously explored in this context [78–85], and can generally be classified into two main categories: randomized algorithms and deterministic algorithms. Randomized algorithms rely on probabilistic decisions made by individual nodes. Each node uses randomization to determine its actions (e.g., whether to join the *MIS* or not). In contrast, deterministic algorithms make decisions based on fixed rules without involving any randomness. Each node follows a predetermined process to decide whether to join the *MIS*.

3.3.1.1 Randomized algorithms

The best-known randomized algorithm, proposed by Luby's algorithm [80] in 1986, correctly constructs a *maximal independent set* of a graph $G(V, E)$ in $O(log\, n)$ time using $O(m\, log\, n)$ messages. In 1984, the author proposed an algorithm to solve the *MISP* [78] in $O((log\, n)^4)$ time using $O((n/(log\, n))^3)$ processors. In 1986, Alon [79] presented another simple parallel randomized algorithm for finding a *maximal independent set* in $O(log\, n)$ time. Two other algorithms, simplified versions of Luby's algorithm with the same complexity, were presented by Erciyes in [86]. In [87], the author introduced in 2010 a randomized distributed *maximal independent set* algorithm for arbitrary graphs in $O(log\, n)$ time with probability $1 - O(n - 1)$, and only needs messages containing 1 bit, and its bit complexity per channel is $O(log\, n)$. The author introduced a Random-priority parallel algorithm, which outperforms its predecessor by ensuring that at least one new node is consistently incorporated into every connected component. In 2016, Ghaffari [88] presented a simple randomized algorithm that achieves near-optimal local complexity for the *MISP*. Their algorithm guarantees that each node terminates within $O(log\, \Delta) + 2^{O(\sqrt{log\ log\, n})}$ rounds. Later, Ghaffari [89] introduced in 2019 a randomized distributed *MIS* algorithm using $O(logn)$-bit messages, achieving a round complexity of min $(log\, \Delta.2^{O(\sqrt{log\ log\, n})}, O(log\, \Delta.log \log\ n) + 2^{O(\sqrt{log\ log n.log \log\ log\, n})})$.

3.3.1.2 Deterministic algorithms

To the best of our knowledge, the first deterministic algorithm for the *MISP* was proposed by Awerbuch et al. in 1989 [90]. Their approach utilizes the concept of network decompositions, yielding a deterministic *MIS* algorithm with a runtime of $O(f(n)^c)$, where c is a constant, and $f(n) = n^{\sqrt{log(log\, n)/log\, n}}$, which allows for the computation of an $O(f(n)^c)$-decomposition. The fastest known deterministic algorithm for *MISP* was introduced by Panconesi and Srinivasan in 1992 [91]. Their method, based on a $(g(n)^d, g(n))$-decomposition, achieves a runtime of $O(g(n)^d)$, where $g(n) = n^{\sqrt{1/logn}}$ and d is a constant. In 2001, Panconesi et al. [92] introduced an elegant deterministic distributed algorithm for computing a *maximal independent set*, demonstrating that it can be found in $O(log^*\, n + \Delta^2)$ deterministic rounds. In 2005, Kuhn et al. [93] proposed a deterministic algorithm that computes a *maximal independent set* in time $O(log\, \Delta.log^*\, n)$ for graphs with bounded growth. In 2012, Blelloch et al. [94] showed that it is possible to obtain a parallel and distributed deterministic algorithm implementation in $O(log^2\, n)$ parallel/distributed rounds.

Despite the reasonable complexity of most distributed *MISP* algorithms cited above, their applicability to unreliable platforms, such as wireless sensor networks (WSNs) and

IoT networks, remains limited, as demonstrated in [95]. Table 3.1 summarizes the most significant *distributed algorithms* to solving the *MISP*.

Table 3.1: Distributed algorithms for MISP

Year	Topology	Method	Complexity	Citation
1984	Arbitrary	Randomized	$O((log\, n)^4)$	[78]
1986	Arbitrary	Randomized	$O(log\, n)$	[79]
1986	Arbitrary	Randomized	$O(log\, n)$	[80]
1989	Arbitrary	Deterministic	$O(f(n)^c)$	[90]
1992	Arbitrary	Deterministic	$O(g(n)^d)$	[91]
2001	Arbitrary	Deterministic	$O(log^*\, n + \Delta^2)$	[92]
2005	Arbitrary	Deterministic	$O(log\Delta.log^*n)$	[93]
2010	Arbitrary	Randomized	$O(log\, n)$	[87]
2012	Arbitrary	Deterministic	$O(log^2\, n)$	[94]
2018	Arbitrary	Randomized	$O(log\, \Delta) + 2^{O(\sqrt{log\ \ log\, n})}$	[88]
2019	Arbitrary	Randomized	min $(log\, \Delta.2^{O(\sqrt{log\ \ log\, n})}$, $O(log\, \Delta.log\, \log\ n)+$ $+2^{O(\sqrt{log\ \ log\, n.log\, \log\ \ log\, n})})$	[89]
Proposed	Arbitrary	Randomized	$O(\Delta + 1)$	/

3.3.2 Existing WBS-based algorithms

In this subsection, we explore existing research that employs the *Wait Before Starting (WBS)* approach. To establish a clear understanding, we first outline the key principles underlying this method.

3.3.2.1 WBS approach

"To the best of our knowledge, the concept of *WBS* was first introduced in the work of [45]. In this concept, each node maintains a set of variables that define its states within the system. A specific crucial variable, denoted by x will be used to determine the order of execution for nodes. The selection of nodes to begin execution will be based on the values of x associated with each node. In most cases, the minimum value of x is considered for the election. Prior to starting the execution of the algorithm, every node must observe a waiting time denoted by wt. The value of the variable wt is calculated using the following formula:

$$wt = x * wtu \tag{3.1}$$

The value of x varies depending on the specific application under consideration. It may represent the identifier, the number of neighbors, the count of private neighbors, the local

energy of the battery, the residual energy, the coordinate in a networkor, or other relevant factors. It can be computed by aggregating the values of other node variables. Meanwhile, the value of *wtu* which represents the waiting time unit, must be set considerably high to facilitate the neighbor selection process and to ensure the reduction of values among the neighbors of the neighbors can be completed successfully. However, if the value to be elected happens to be the maximum, a transformation is applied, specifically:

$$x = c - x \tag{3.2}$$

, where c is a constant greater than or equal to x. This transformation ensures that the maximum value is appropriately handled within the given context.

3.3.2.2 Previous WBS-based algorithms

Bounceur et al. [45] introduced a novel leader election algorithm, termed *WBS*, specifically designed for wireless sensor networks in the context of smart cities and *IoT networks*. The *WBS* algorithm operates by assigning each node a unique waiting time based on its value. The node with the shortest waiting time (smallest value) is elected as the leader. Once elected, the leader initiates its program and broadcasts a message to inform other nodes of its leadership. Simulation results demonstrate that *WBS* significantly outperforms traditional algorithms in terms of energy consumption. Kadjouh et al. [96] proposed an enhanced version of the *WBS* algorithm, tailored for handling real-valued data, such as GPS coordinates, and optimized for large-scale *IoT networks*. Their algorithm maintains the core principles of *WBS* while incorporating improvements for efficiency and fault tolerance. In a more recent work, Kadjouh et al. [97] introduced a dominating tree-based leader election algorithm for smart city *IoT networks*. Their approach leverages the concept of boundary nodes and a flooding process to construct a spanning tree. The leader is determined as the node at the far-left end of the network. Simulation results indicate that this algorithm achieves substantial energy savings, with reductions of up to 85 % for finding the minimum x-coordinate and 30 % for finding the minimum random value. Bezoui et al [71]; studied the domination problem using the *WBS* approach. They proposed a new *distributed algorithm* for finding a dominating set in WSNs and *IoT networks*, which also allows for the determination of a minimal dominating independent set. In their algorithm, nodes with the highest number of neighbors declare themselves first, initiating the process, with their waiting time proportional to the energy consumed. Since nodes with maximum degrees start first, the author used the transformation in formula (3.2). Simulation results on various randomly generated networks show that the proposed algorithm outperforms standard approaches in terms of energy efficiency while

ensuring the accurate identification of the dominating set.

Table 3.2 summarizes *WBS-based algorithms* and their x values, used to calculate waiting times for solving various graph parameters in *IoT networks* using the CupCarbon simulator.

Table 3.2: Previous WBS algorithms

Year	Output	Topology	value of x	Citation
2017	Leader	Arbitrary	node value	[45]
2017	MDS	Arbitrary	$n - d(v)$	[71]
2019	Leader	Arbitrary	GPS-coordinates	[96]
2023	Leader	Arbitrary	local minimum value	[97]
Proposed	MIS	Arbitrary	$d(v)$	/

3.4 A distributed algorithm for MIS

In this section, we introduce the proposed algorithm, named *WBS-MIS*, based on the *Wait Before Starting* concept. It is designed for both anonymous and non-anonymous *IoT networks*. We begin by presenting all the variables and functions that will be utilized in our algorithm. Following this, we provide a description of the algorithm. Then, we discuss its convergence complexity, followed by an illustrated example.

3.4.1 Variables and primitive functions

All the main variables and primitive functions used by *WBS-MIS* algorithm are described in Tables 3.3 and 1.2, respectively. Two additional functions will be utilized, each de-

Table 3.3: Description of the variables

Variables	Description
x	Used to store de degree of a node
mis	Boolean variable that indicates whether or not the node in the maximal independent set
$rfmis$	Boolean variable. It takes the value true when it receives a first message from an independent node
$rfnomis$	Boolean variable. It takes the value true when it receives a first message from a non independent node
ts	Used to store the start-up time of the current node
tf	Used to store the final time of the initial waiting time of current node
nwt	Used to store the waiting time associated with the updated count of neighbors.

scribed as follows:

- *rand(a, b):* Generates an integer random value between a and b exclusive;
- *getT ime()*: Returns the local time of a node.

3.4.2 Description of WBS-MIS

In this subsection, we give a detailed description of our proposed algorithm. We present a flowchart (see Figure. 3.1) summarizing the computational method, and provide the algorithm's pseudo-code (see Algorithm 3.1). Following this, we describe the various steps that compose the computation.

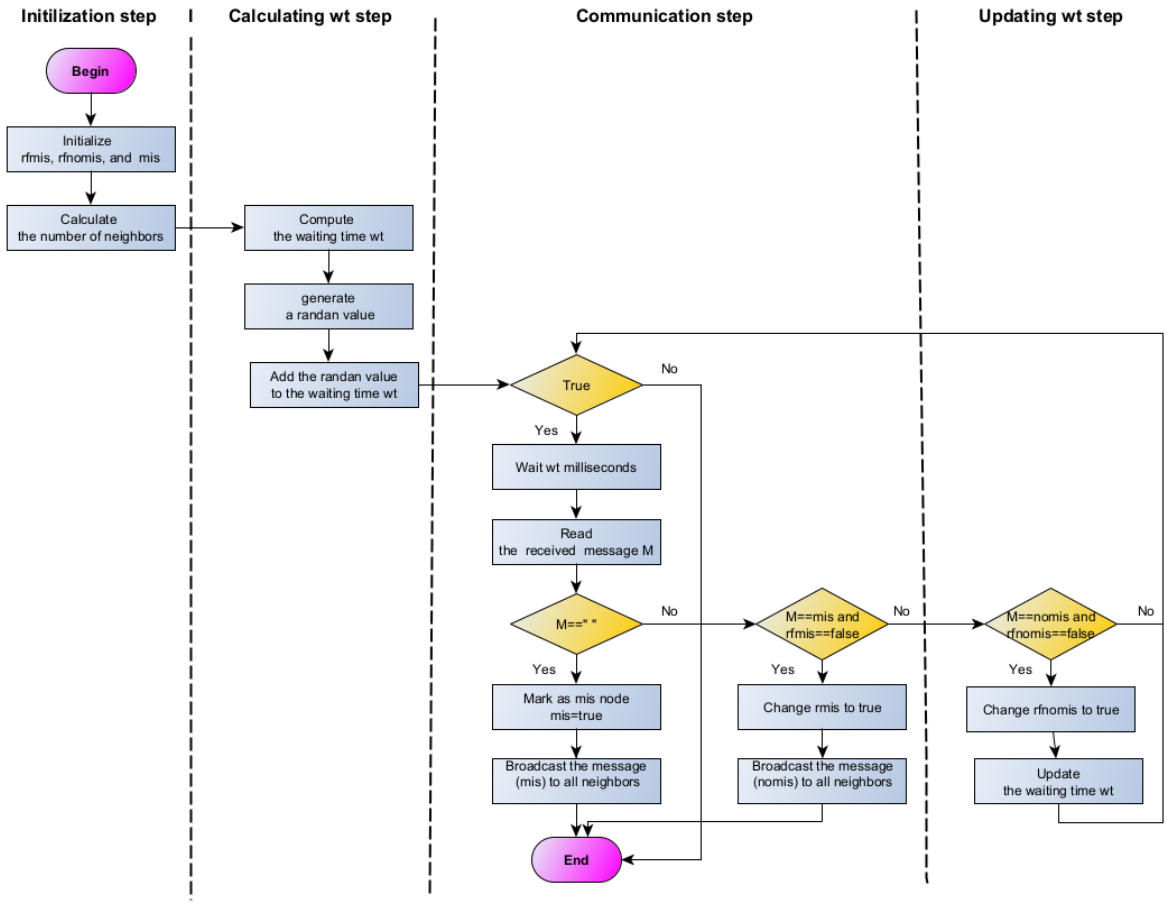


Figure 3.1: Flow diagram for the *WBS-MIS* algorithm.

The algorithm uses the main variables and primitive functions which are described in Tables 3.3 and 1.2. It requires, as input data, the values of the waiting time unit *wtu* and

Algorithm 3.1: WBS-MIS

```
Data: wtu, ub
Result: mis
begin
    mis ← false;
    rfmis ← false;
    rfnomis ← false;
    x ← getNNeig();
    wt ← x * wtu + rand(1, ub);
    ts ← getTime();
    tf ← ts + wt;
    while (true) do
        message ← read(wt);
        t ← getTime();
        if (message = null) then
            mis ← true;
            send(mis, *);
            stop();
        else
            if (message = mis and rfmis = false) then
                rfmis ← true;
                send(nonmis, *);
                stop();
            if (message = nomis and rfnomis = false) then
                rfnomis ← true;
                nwt ← (x − 1) * wtu + rand(1, ub);
                wt ← (nwt − wt) + (tf − t);
```

the upper bound ub which will be used to generate a random value, and as output, the boolean mis value. Our algorithm can be divided into four main steps:

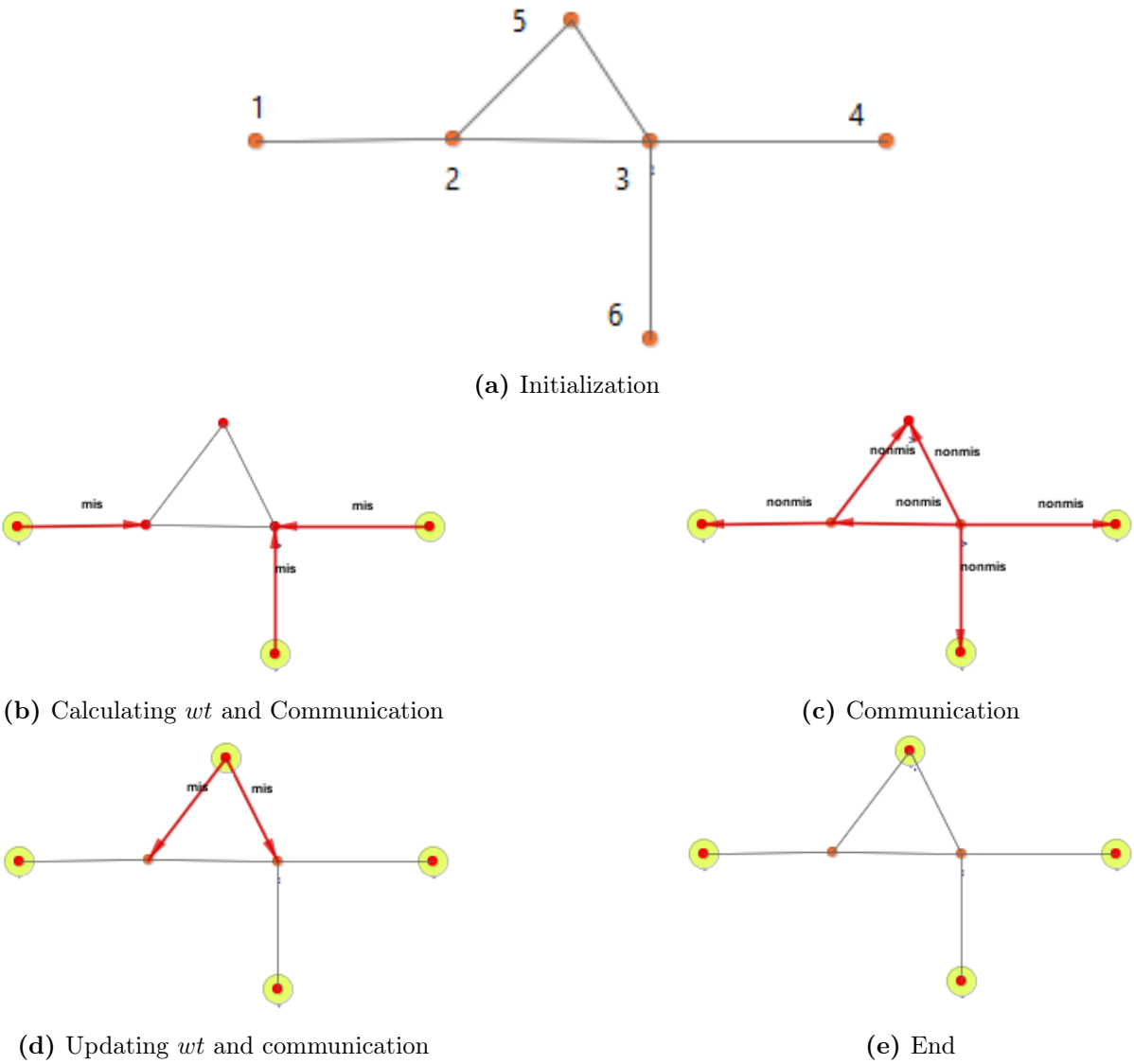
- **Initialization step:** in this initial step, each node, after obtaining the number of its neighbors, initializes the boolean values: $rfmis, rfnomis$, and mis, setting them to False. This signifies that the node is not considered a mis node and has not sent or received any messages up to this point (see lines 1:5).
- **Calculating wt step:** each node in the network calculates its waiting time, which depends on its degree as well as the chosen unit of time wtu. The latter must be sufficiently high to allow the process of selecting the neighbors and decreasing the values of the neighbor of the neighbors to finish. (see line 6)
- **Communication step:** after waiting for wt milliseconds, if the current node has

not received any messages, it will be marked as a mis node. It must inform its neighbors by sending the message *mis*, and then it stops execution and becomes inactive. In the case that there is a node with a shorter waiting time than the current node, and it has already sent the message *mis* to inform the current node, the current node sends the message *nomis* to notify its neighbors that it remains a *non mis node* and becomes inactive too (see lines 10:20).

- **Updating** *wt* **step:** if, after a waiting time less than *wt*, the node receives the message *nomis*. This means that the neighbors of the current node are both not included in the *Maximal Independent Set* and are disabled too, then it is necessary to update their number of neighbors. The waiting time must be updated too since it depends on the number of neighbors (see lines 21:24)

3.4.3 Illustrated example

We illustrate with an example given in Figure. 3.2 how the proposed algorithm *WBS-MIS* constructs a *Maximal Independent Set*. In this example, we consider a network of six nodes with identifiers 1, 2, 3, 4, 5, and 6 (note that node identifiers are used only to simplify the explanation). The initial state is shown in Figure 3.2a with all nodes neither marked nor included in the *Maximal Independent Set*. Let's assume that the value of *wtu* is equal to 50 milliseconds and the value of *up* is equal to 10. Initially, nodes 1, 4, and 6 will wait for 50, 53, and 58 milliseconds, respectively. Consequently, they will be the first to start sending messages *mis* after marking themselves as *MIS nodes*, owing to their lower degrees in the network, as illustrated in Figure 3.2b. And the estimated waiting times for nodes 2, 3, and 5 are 151, 204, and 107 milliseconds respectively. Next, during the waiting period, nodes 2 and 3 receive messages sent from nodes 1, 4, and 6. Consequently, they stop execution without marking after transmitting *nomis* messages to node 5, as illustrated in Figure 3.2c. Then, Node 5 has received the *nomis* messages from nodes 2 and 3 and updates its waiting time *wt*. Subsequently, it waits for approximately 230 milliseconds before marking itself, transmitting the *mis* message, and halting its execution.



(a) Initialization

(b) Calculating wt and Communication

(c) Communication

(d) Updating wt and communication

(e) End

Figure 3.2: Illustration of the *WBS-MIS* algorithm.

3.4.4 Complexity

In this subsection, we will study the complexity of the proposed algorithm *WBS-MIS* in terms of the number of iterations needed for execution, as well as the number of sending messages.

Proposition 3.1. *The WBS-MIS algorithm finds a maximal independent set of nodes in a given IoT network after* $(\Delta + 1)$ *iterations.*

Proof. Consider an *IoT network* modeled as a connected undirected graph $G = (V, E)$ with n nodes and m edges. we'll use the notations we've seen before (see Section. 2.2).

Each node $v \in V$ has $d(v)$ neighbors, were $1 \leq d(v) \leq \Delta$. Nodes with minimum degrees ($d(v) = 1$ in the best case) initiate the execution by distributing a message to all neighbors and then halt the execution. Therefore, it is clear that this first-type of nodes requires only one iteration, as they do not need any information except the waiting time. The neighbors of these first-type nodes, considered the second-type, and having higher degrees than them, will be informed by the first-type nodes before their waiting time elapses. Consequently, the second-type nodes broadcast messages to the waiting nodes, categorized as the third-type. It is also evident that these second-type nodes require only one iteration. The nodes of the third-type received messages from all the neighboring nodes of the second-type, and for each message received, they updated their waiting times in each iteration. So, it's obvious that the worst-case number of iterations for this third-type of nodes equals the maximum degree in the network Δ, plus an additional iteration to calculate the waiting time at the start of execution. □

Proposition 3.2. *In an IoT network, The WBS-MIS algorithm finds a maximal independent set of nodes using a total number of messages sent equal to n.*

Proof. As we have mentioned in Proposition 3.1, there are three types of nodes in the IoT network. We denote the number of nodes of the first-type as K, the number of nodes of the second-type as L, and the number of nodes of the third-type as P, all of which are less than n. K nodes of the first-type broadcast a single message to all their neighbors. Therefore, K messages are sent. Among the remaining $(n-K)$ nodes, L nodes considered as second-type broadcast a single message to all their neighbors. Thus, L messages are sent by these nodes. The remaining $(n - K - L)$ nodes, considered as the third-type, send $(n - K - L)$messages. Adding up the messages sent by each type of node, we get $(K + L + (n - K - L) = n)$. Therefore, the total number of messages sent is indeed n. □

3.5 Conclusion

In this chapter, we introduced the second *distributed algorithm* of our approach, focusing on *MISP*. We provided the necessary theoretical definitions and formulated the problem statement to ensure a comprehensive understanding of the *MISP*. To contextualize our work, we presented a state-of-the-art overview, categorizing and critically reviewing the algorithms previously proposed in the literature to address this issue. Additionally, we outlined the methodology used in the design of our algorithm and examined relevant research in the field. Finally, we introduced our novel distributed MIS algorithm, offering an in-depth description of its structure and elaborating on the various stages involved in its implementation.

In the next chapter, we will focus on the *critical node detection problem*, which represents the third key contribution of this thesis. We will introduce distributed solutions designed to effectively address this challenge.

Chapter 4

Distributed Identification of Critical Nodes in Tree-Based IoT Networks

4.1 Introduction

One of the most effective approaches to enhancing *IoT* security is *critical node detection*, the process of identifying key nodes whose failure or compromise could significantly disrupt the network. The problem of finding these nodes is known as the *Critical Node Detection Problem (CNDP)*. This methodology is essential to strengthening network resilience and mitigating potential security threats. Conversely, tree structures offer notable benefits for *IoT networks*, including enhanced energy efficiency, scalability, simplified management, fault tolerance, and improved data handling capabilities. These characteristics render tree topologies an ideal selection for various IoT applications, guaranteeing dependable and efficient network performance.

This chapter discusses the *3C-CNP*, a specialized variant of the well-known *CNDP* in *IoT networks*. We present a Distributed Algorithm for the Component-Cardinality-Constrained Critical Node Problem *(DA3C-CNP)*, aimed at identifying *critical nodes* within tree-structured *IoT networks*. The chapter begins with a comprehensive literature review on *CNDP*, providing an overview of existing approaches and methodologies. We then formally define the *3C-CNP*, discuss its practical applications, and review related work in the field. Following this, we introduce the proposed *DA3C-CNP* algorithm, designed to function in both anonymous (identifier-free) and non-anonymous *IoT networks*. Finally, we conclude the chapter by summarizing key findings and highlighting potential future research directions.

4.2 Literature review on CNDP

The *CNDP* aims to identify a subset of nodes in graphs or networks whose removal maximally disrupts network connectivity based on specific predefined metrics. The *CNDP* is typically defined as follows: The problem inputs a graph $G = (V, E)$ and a predefined connectivity metric σ, and outputs a set of nodes $S \subseteq V$ whose removal optimizes the objective function $f(\sigma)$. This issue has applications across various domains, such as social network analysis [98, 99], network immunization [100], transportation engineering [101, 102], telecommunications [103], military strategic planning [104], IoT network analysis [14], and network security [105], among others. The identification of critical nodes in network analysis is directly relevant to network security. Consequently, *CNDP* has garnered significant attention in recent years, with numerous connectivity metrics and variations of this problem being defined and examined in the literature. These variants depend on how the network becomes disconnected following the removal of nodes, and include: the Critical Node Problem (CNP) [106, 107]; the problem of Maximizing the Number of Connected Components (MaxNum) [108, 109]; the problem of Minimizing the Size of the Largest Connected Component (MinMaxC) [110, 111]; the β-Vertex Disruptor Problem, where β is a given threshold [112, 113]; and the Component-Cardinality-Constrained Critical Node Problem (3C-CNP) [114, 115], among others. It is established that the majority of CNDP variants are NP-hard, even when limited to specific graph classes.

CNDP has its origins in the research conducted by Borgatti et al. [98], who introduced various methods and metrics for identifying the most significant nodes in a network, referred to as key players. Arulselvan et al. [99] expanded upon this foundation by concentrating on the pairwise connectivity metric, formally defining the critical node problem as the identification of a set of nodes whose removal minimizes pairwise connectivity, or path survivability, within the network. The authors established the NP-completeness of the recognition version of *CNDP* on general graphs, formulated a linear programming model for effective problem-solving, and introduced a greedy algorithm that surpassed other methods on randomly generated instances. Di-Summa et al. [107] subsequently presented complexity results and developed algorithms for different versions of the *CNDP* on trees, including edge costs and node weights. The authors established that the *CNDP* on trees retains NP-completeness under specified general connection costs. They proposed a polynomial-time algorithm utilizing a dynamic programming approach for instances involving unit connection costs. Addis et al. [116] established the NP-completeness of the *CNDP* across various graph classes, such as split graphs, bipartite graphs, and complement bipartite graphs. They proposed a polynomial-time algorithm for identifying critical

nodes in graphs characterized by bounded treewidth. Shen et al. [117] established the NP-completeness of the critical node problem for Unit-Disk graphs and Power-Law graphs. The authors proposed efficient greedy algorithms for identifying critical nodes and links, with results utilized to evaluate network vulnerability. Guettiche and Kheddouci [102] examined the issue of critical nodes and links, employing the shortest path metric. The authors proposed algorithms aimed at evaluating the reliability of transportation networks, thereby broadening the applicability of *CNDP* methodologies in infrastructure analysis. Shen et al. [109] examined novel variants of the *CNDP*, focusing on two specific metrics: the maximization of connected components and the minimization of the largest component's size. The authors proposed polynomial-time algorithms for addressing these variants on series-parallel graphs, k-hole graphs, and trees. Lalou et al. [115] investigated the critical node problem concerning the limitation of the largest connected component's size and proposed a novel variant known as the *Component-Cardinality-Constrained Critical Node Problem (3C-CNP)*. For an extensive examination of *CNDP* variants and their applications, readers should consult the survey conducted by Lalou et al. [22]. Recently, several research works have been published, such as Hosteins et al. [118], which required that a critical set of nodes be connected. In contrast, Di-Summa and Faruk [119] initiated the hybrid problem of critical node/edge detection, which aims to identify a set of nodes and/or edges of specified cardinality, whose removal maximally disrupts network connectivity through pairwise minimization. A recent survey on critical node detection, including its applications and challenges, was conducted in 2023 [120].

Identifying critical nodes in *IoT networks* is essential for improving the reliability and resilience of communication systems. This acknowledgment forms the basis for multiple investigations concerning the *CNDP*. In [121], the authors investigated the CNDP to enhance network defense in MANET-IoT networks. A method called Dynamic Critical Node Identification (DCNI) was proposed for the identification of critical nodes. The proposed DCNI exhibits a complexity of approximately $O(m * n^2)$, with n denoting the number of nodes and m indicating the number of links in the network. The authors examine the *CNDP* in Industrial Wireless Sensor and IoT networks in [122]. A two-phase algorithm is proposed: Phase I utilizes a distributed method for critical node detection (Algorithm 1), and Phase II improves node resilience via a centralized method (Algorithm 2). The proposed algorithms necessitate $O(log(n))$ time for convergence and $O(\delta(logn))$ for Critical Node detection, where n denotes the number of IoT devices and δ signifies the cost associated with message forwarding. Recently, Ishfaq et al. [123] have examined the *CNDP* within an IoT network. An efficient and minimalistic integer linear programming solution was developed to optimize the cost of a network attack. The experimental results indicate that this approach is efficient and scalable, making it appropriate for large-scale

infrastructure systems. The solution fulfills one of two objectives: either maximizing the damage inflicted on the network within a specified budget or minimizing the cost of an attack that results in a predetermined level of damage. Ugurlu et al. [14] provide a survey of *critical node detection* methods in IoT, detailing their applications and associated challenges.

4.3 3C-CNP variant

In this section, we provide a comprehensive definition of *3C-CNP*, discuss its real-world applications, and review existing literature related to this variant.

4.3.1 Problem definition

Let $G = (V, E)$ be an undirected, unweighted, and connected graph representing an *IoT network*, where V denotes the set of n vertices (or nodes) and $E \subseteq V \times V$ represents the set of m edges. For a subset of nodes $S \subseteq V$, the subgraph induced by S is denoted as $G[S]$, which contains the vertices in S and all edges in E connecting vertices in S. Conversely, $G[V - S]$ represents the subgraph induced by the set of vertices $V - S$, consisting of the vertices in V that are not in S and the edges between them.

The *Component-Cardinality-Constrained Critical Node Problem (3C-CNP)* seeks to find the minimal subset of nodes within a graph G, the deletion of which results in a set of connected components of cardinality at most B each one, where B is a given integer bound. Formally, the *3C-CNP* takes as input a graph $G = (V, E)$, and returns as output a set of nodes $S \subseteq V$:

Component-Cardinality-Constrained Critical Node Problem (3C-CNP)
Input: A graph $G(V, E)$ and an integer B.
Output: A minimum set of nodes $S \subseteq V$, such that $|h| \leq B$ for each subset of connected nodes $h \in G[V - S]$.

4.3.2 Applications

The *3C-CNP* has emerged as a prominent area of research owing to its diverse applications in multiple fields, such as social network analysis, biology, communication networks, and network reliability. This subsection presents the key applications identified in the literature, as depicted in Figure 4.1.

- **Social network analysis:** it involves the examination of social structures through the use of networks and graph theory. It focuses on the relationships and interactions among individuals or groups, providing insights into the dynamics of social systems. Community detection is a fundamental aspect of social network analysis [124, 125]. For example, specifying a threshold for community size allows for the identification of network communities accordingly. This is especially pertinent for terrorist networks, in which the sizes of terrorist groups or communities may be identifiable. In these scenarios, critical nodes serve as connections between distinct communities. Consequently, addressing the *3C-CNP* facilitates the identification of these terrorist organizations.

- **Computational Biology:** biological organisms are composed of interconnected proteins that interact to form a protein interaction network, which can be represented as a graph, where nodes correspond to proteins and edges represent their interactions. The *3C-CNP* methodology is capable of identifying the minimal set of proteins whose removal would effectively neutralize the harmful organism [126].

- **Immunization:** in cases of adverse events, including virus transmission in networks, epidemics, or malware dissemination, *3C-CNP* assists in identifying individuals (or devices) whose immunization can halt the epidemic's spread. Furthermore, it aids in identifying the origin of the diffusion. [127, 128]

- **Network reliability:** the security of network applications against malicious behavior is a fundamental design consideration. The implementation of the *3C-CNP* enhances network security by offering a metric for assessing network vulnerability. The *3C-CNP* evaluates vulnerability by the principle that a network necessitating the removal of more critical nodes for partitioning is deemed less vulnerable. The removal of fewer nodes facilitates easier network compromise [115, 129].

4.3.3 Related works

In Section 4.2, a comprehensive overview of research pertaining to *CNDP* in general has been presented. This subsection delves into a detailed examination of studies and methodologies that specifically focus on the *3C-CNP* variant.

The *3C-CNP* is a variant derived from the Cardinality-Constrained Critical Node Problem (CC-CNP), originally introduced by Arulselvan et al. in 2011 [130] to identify critical nodes in telecommunication networks. Their work focuses on minimizing the number of vertices whose removal causes the network to become disconnected, with the

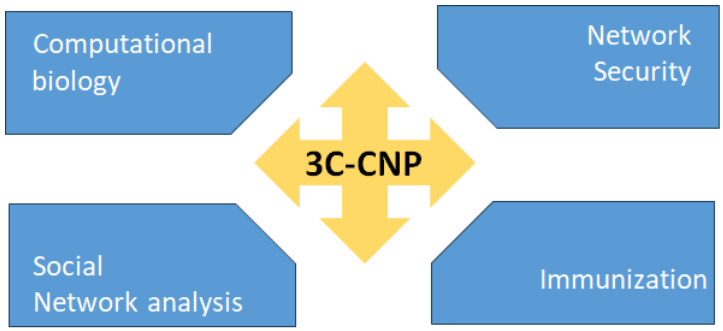


Figure 4.1: Examples of *3C-CNP* applications.

removal constrained by a predefined cardinality. They also established that the CC-CNP is NP-complete for general graphs. Building upon this foundation, Lalou et al. [115] introduced and explored a new variant, called *3C-CNP*. In this version, the goal is to identify a minimal set of nodes whose removal guarantees that the size of each connected component in the resulting graph remains within a specified bound. Lalou et al. further demonstrated the NP-completeness of the recognition version of the problem, both for general graphs and for bounded-degree graphs with a maximum degree of $\Delta = 4$

A variety of centralized methods for solving the *3C-CNP* on general graphs have been proposed in the literature, including greedy algorithms [103, 114, 131], integer programming [114, 132], evolutionary algorithms [112, 114, 133], and approximation methods [132, 134]. When considering specific graph classes, the *3C-CNP* has been shown to be NP-complete on split graphs and trees, where nodes and connections have nonnegative weights and costs, respectively [115, 129]. The study in [115] examined the *3C-CNP* problem on proper interval graphs and trees. For proper interval graphs, they proposed a polynomial-time algorithm with a time complexity of $O(n^2)$. For trees, they developed an algorithm with a time complexity of $O(n)$ for unweighted trees and $O(n^2)$ for weighted trees. In [124], the authors introduced a dynamic programming algorithm that solves the *3C-CNP* in polynomial time and space for bipartite permutation graphs, with a time complexity of $O(nB^2)$. More recently, in [135], a polynomial-time algorithm was presented for solving the *3C-CNP* on chordal graphs with a maximum node degree of $\Delta = 3$. This algorithm computes an exact solution efficiently with a time complexity of $O(n^2)$. As explained in Subsection 4.4.4, we present a distributed algorithm in this thesis with a time complexity of $O(n - l)$, where l is the number of leaf nodes. Table 4.1 provides a summary of the most notable approaches to solving the *3C-CNP* problem.

As far as we are aware, no *distributed algorithms* exist for solving *3C-CNP* in either general or particular graph classes, and no research has yet to apply the *3C-CNP* to *IoT networks*.

Table 4.1: An overview of the 3C-CNP algorithmic solutions

Citation / Year	Topology	Complexity class	Solution	Complexity
[130] (2011)	Arbitrary	NP-complete	Centralized	$O(n^2 + nm)$
[115] (2016)	Arbitrary with $\Delta \leq 4$	NP-complete	/	/
[115] (2016)	Trees	Linear	Centralized	$O(n)$
[115] (2016)	Weighted trees	Polynomial	Centralized	$O(n^2)$
[115] (2016)	Proper interval	Polynomial	Centralized	$O(n^2)$
[124] (2019)	Bipartite permutation	Polynomial	Centralized	$O(nB^2)$
[129] (2023)	Split	Polynomial	/	/
[135] (2024)	Chordal	Polynomial	Centralized	$O(n^2)$
Proposed	Tree	Linear	Distributed	$O(n - l)$

4.4 A distributed algorithm for 3C-CNP

The suggested algorithm, *DA3C-CNP* (Distributed Algorithm for the Cardinality-Constrained Critical Node Problem on Trees), is presented in this section. First, we define the functions and variables used in the algorithm. A thorough explanation of how the algorithm works is then given. Lastly, we provide an example to show how it is implemented.

4.4.1 Primitive functions and variables

Tables 4.2 and 1.2, respectively, provide descriptions of all the key variables and basic functions that our algorithm uses.

Table 4.2: Description of the variables

Variables	Description
d	Used to store the degree of a node
$Critical$	Boolean variable that indicates whether or not the node in the critical nodes set
S	An integer variable used to store the cumulative sum of message values received by the current node.
NR	An integer variable used to track the count of messages received by the current node from its neighboring nodes.

4.4.2 Description of DA3C-CNP

This subsection gives the *DA3C-CNP* pseudocode and a detailed description of how it is executed. Figure 4.2 displays the flowchart for the algorithm. Every network node

Algorithm 4.1: DA3C-CNP

```
Data: B < n/2
Result: Critical
begin
    Critical ← false;
    S ← 1;
    NR ← 1;
    d ← getNNeig();
    if (d == 1) then
        send(1, ∗);
        stop();
    else
        while (true) do
            M ← read();
            S ← S + M;
            NR ← SR + 1;
            if (S > B) then
                Critical ← true;
                send(0, ∗);
                stop();
            else
                if (NR == d) then
                    send(S, ∗);
                    stop();
```

executes an instance of *DA3C-CNP*. For each node, it generates a Boolean output for the variable $Critical$ after receiving the upper bound value B as input. The DA3C-CNP first initializes the variable S to 1 and the value of NR (number of received messages) to 1 for every node, setting $Critical$ to false. The function $getNNeig()$ is used to determine the current node's degree. Leaf nodes start the execution by broadcasting a message with the value 1 right away, after which they stop. Conversely, non-leaf nodes keep getting messages from their neighbors. Each time a message is received, the node increments the value of NR and adds the received message values to S. If, during execution, the cumulative sum S at a node exceeds the upper bound B, the node is immediately included in the critical nodes set. The node then sets its $Critical$ variable to true, broadcasts a message with a value of 0, and halts further execution. If S remains less than or equal to

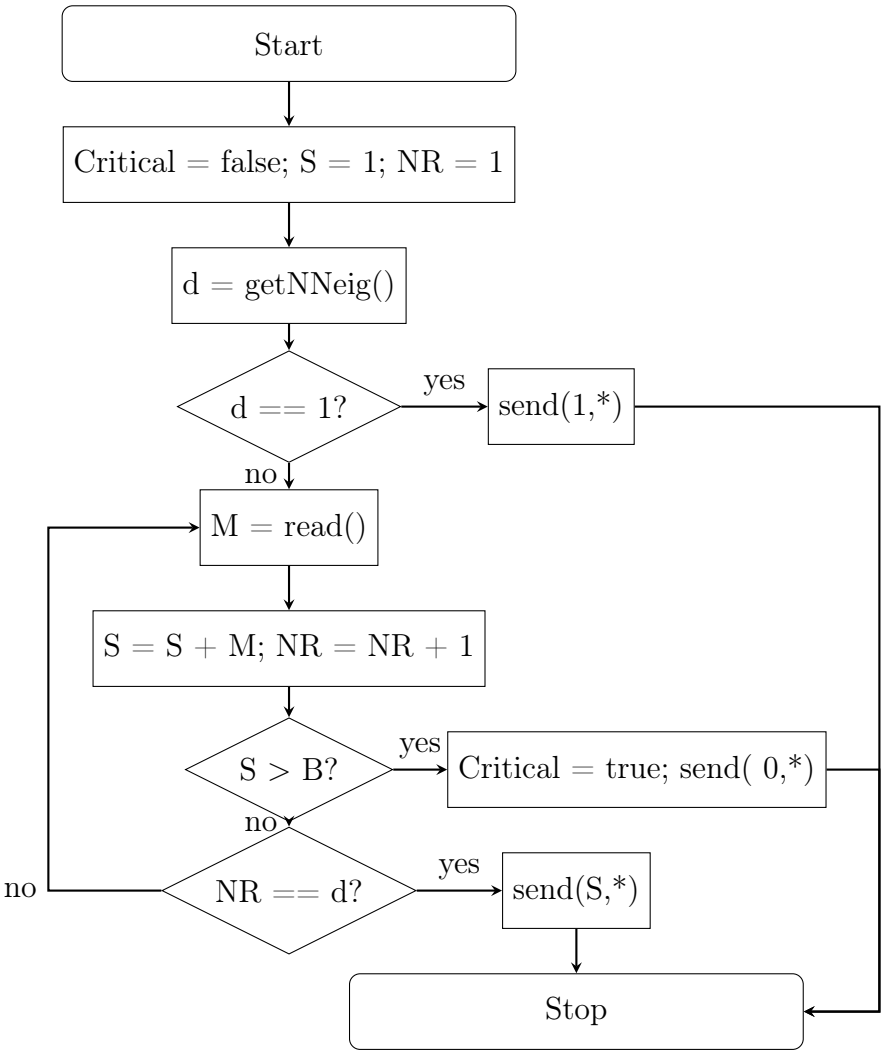


Figure 4.2: *DA3C-CNP* FlowChart.

B, the node awaits the receipt of all messages from its neighbors before broadcasting its updated S value. This approach ensures that nodes fully incorporate the contributions from all neighboring nodes prior to finalizing their status.

4.4.3 Illustrated example

We use the tree network shown in Figure 4.3a to demonstrate how Algorithm *DA3C-CNP* works. Figure 4.3f displays the result of this algorithm on the provided graph. As explained below, Algorithm *DA3C-CNP* functions in this network in five steps. As illustrated in Figure 4.3b, the leaf nodes (5, 6, 9, and 10) first broadcast a message to their parent nodes with a value of 0 and stop their programs from running. After that, nodes 5 and 6 send two messages with a value of 1 to non-leaf node 3. S=3 is the result of adding the received values. As shown in Figure 4.3c, node 3 broadcasts a message with a

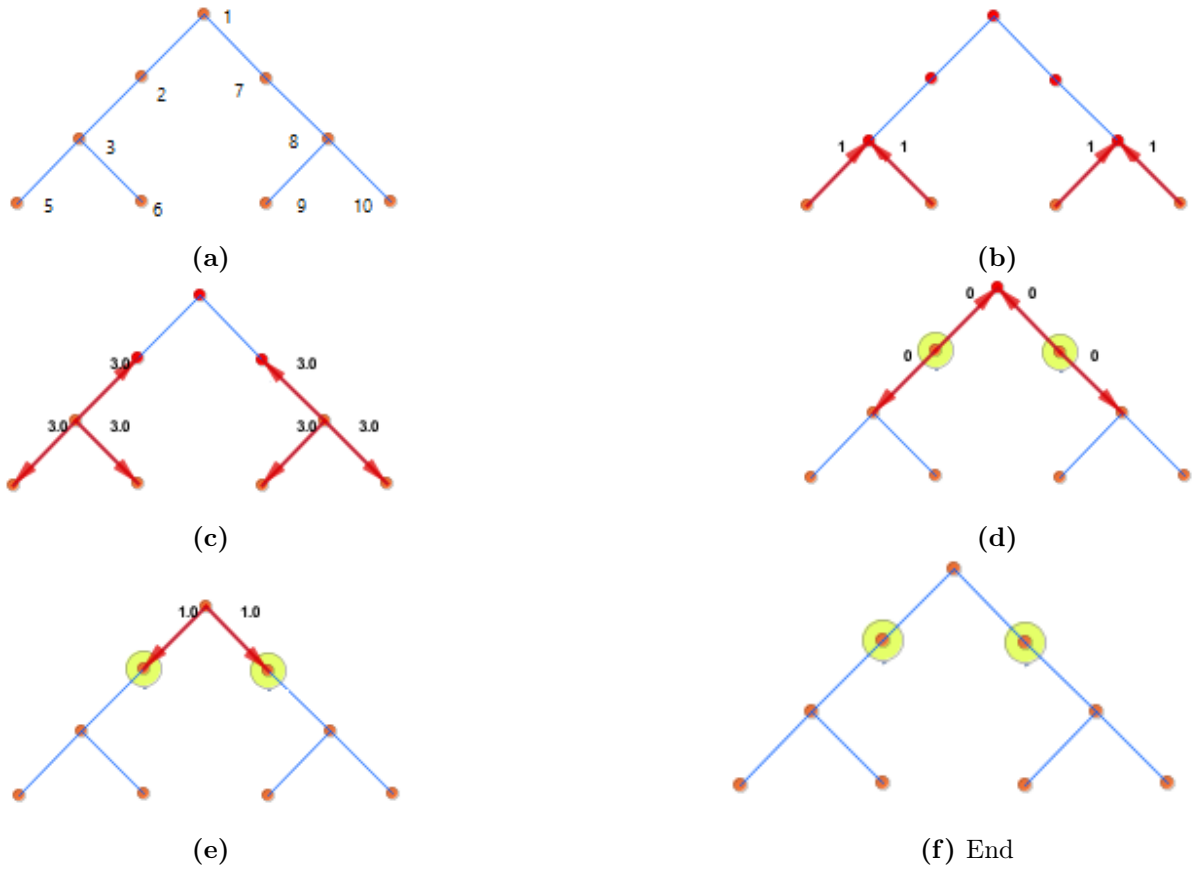


Figure 4.3: Depiction of *3C-CNP* within a tree *IoT network* with $B = 3$.

value of 3 because the sum is not greater than the upper bound B=3. Similarly, non-leaf node 8 receives two messages with a value of 1 from nodes 9 and 10, respectively, and sums them to S=3. As the sum is not greater than 3 B=3, node 8 broadcasts a message with a value of 3 (see Figure 4.3c). Then, two messages with a value of three are sent to nodes 7 and 2, respectively. Both nodes 2 and 7 are designated as critical since they update their internal variable S, and the updated values are greater than the upper bound B. As shown in Figure 4.3d, they broadcast messages with a value of 0 and stop their programs from running. Ultimately, node 1 receives two messages from nodes 2 and 3, each of which contains a value of 0. It halts the execution of its program and increases its variable S (refer to Figure 4.3e). In conclusion, the execution of Algorithm DA3C-CNP on the specified network follows a five-step process, as illustrated in Figures 4.3a through 4.3f.

4.4.4 Complexity

This subsection examines the complexity of the *DA3C-CNP* algorithm, concentrating on two primary metrics: the iterations necessary for execution and the messages exchanged during its operation. We present three primary propositions along with their respective proofs.

Proposition 4.1. *The DA3C-CNP algorithm determines a minimal set of critical nodes in a tree-structured IoT network using* $(\Delta - 1)$ *iterations for each node.*

Proof. Consider an *IoT network* represented as an undirected connected graph $G = (V, E)$ that constitutes a tree, where V denotes the set of n nodes and E signifies the set of m edges. Every node $v \in V$ possesses a degree $d(v)$, constrained by $1 \leq d(v) \leq \Delta$, where Δ denotes the graph's maximum degree.

The *DA3C-CNP* algorithm starts by processing the leaf nodes, which are characterized by a degree of $d(v) = 1$. In this phase, each leaf node performs a single iteration, transmitting a message with value 1 to its parent node before terminating. Consequently, leaf nodes complete their task in just one iteration. For nodes that are not leaves, the execution proceeds as follows:

1. A non-leaf node awaits messages from all its neighboring nodes, excluding its parent node. Upon receipt of a message, it updates its message count (NR) and increases its cumulative sum (S) according to the values of the acquired messages.

2. If the sum S surpasses the specified threshold B, the node is classified as a critical node. It transmits a message with a value of 0 and ends its execution.

3. If $S \leq B$, the node awaits the receipt of messages from all its neighbors, excluding the parent. Upon receiving all messages, it transmits its cumulative sum S to its parent node and complete its operation.

In the most unfavorable scenario, each non-leaf node awaits messages from all its descendant nodes in the tree. Given that a node can possess a maximum of $\Delta - 1$ child nodes (with the exception of the parent node in the tree topology), the upper limit of iterations necessary for a non-leaf node to finalize its execution is $\Delta - 1$.

The hierarchical structure of trees allows execution to progress incrementally from leaf nodes to the root node. The total number of iterations necessary for the *DA3C-CNP* algorithm to find all critical nodes in the tree is limited to $\Delta - 1$ □

Proposition 4.2. *In an IoT network, the DA3C-CNP algorithm finds a minimal set of critical nodes using a total of n messages.*

Proof. The tree structure guarantees that there are precisely l leaf nodes, and $n-l$ non-leaf nodes, including the root.

1. By definition, leaf nodes possess a degree of 1. In the implementation of the DA3C-CNP algorithm, each leaf node transmits a singular message to its parent node within the tree structure prior to finishing execution. Consequently, the aggregate quantity of messages transmitted by the leaf nodes is $l * 1$.

2. Non-leaf nodes await messages from all their child nodes prior to executing any actions. Each non-leaf node integrates the received information into a singular message, which it transmits to its parent node in the hierarchy. This guarantees that each non-leaf node transmits precisely one message during its operation. Consequently, the aggregate quantity of messages transmitted by the no-leaf nodes is $(n - l) * 1$.

The algorithm requires a singular upward communication flow from the leaf nodes to the root, with each node transmitting a maximum of one message; consequently, the overall amount of messages exchanged in the network is corresponding to the total number of nodes, n $((l * 1) + (n - l) * 1)$. This incorporates both leaf nodes and non-leaf nodes, as each node engages precisely once in message propagation. □

Proposition 4.3. *In a tree-structured IoT network, the DA3C-CNP algorithm finds a minimal set of critical nodes using a number of iterations less than $n - l$.*

Proof. Every leaf node operates its program autonomously and simultaneously. Upon execution, it transmits a singular message (value 1) to its parent node and subsequently terminates. This procedure involves one iteration for all terminal nodes. Non-leaf nodes, however, collect messages from all their subordinate nodes. Upon receiving a message, the non-leaf node increases its cumulative value (S) and increases its number of messages obtained (NR).

The algorithm operates in a level-by-level manner, with execution propagating from the leaf nodes to the root. The total number of iterations is determined by the number of levels in the tree. In the most adverse scenario, every level necessitates the execution of all non-terminal nodes at that level. Considering that there are $n - l$ non-leaf nodes in the tree, the total number of iterations necessary to determine the minimal set of critical nodes is unequivocally less than $n - l$.

□

4.5 Conclusion

This chapter has provided a comprehensive overview of the *3C-CNP* and its significance within the context of general graphs and networks. It included a detailed review of the existing literature on the *CNDP*, encompassing its various approaches, algorithms, and applicability across different network structures. We introduced the proposed *DA3C-CNP* for Trees, highlighting its capability to function effectively in both anonymous (identifier-free) and non-anonymous *IoT network* environments. A detailed description of the algorithm was presented, covering key variables, functions, and the operational mechanisms involved. To illustrate its practical application, an example was provided to demonstrate the algorithm's execution and its potential to effectively address the *3C-CNP* within *IoT* frameworks. Furthermore, the chapter included an analysis of the computational complexity of the *DA3C-CNP*, providing insights into its efficiency and scalability.

In the subsequent chapter, we will implement the proposed algorithms that constitute our approach, leveraging a simulation tool to validate their performance and compare them with existing results from similar problems. This simulation-driven approach will allow for a thorough and systematic performance evaluation of the algorithms in the context of representative *IoT* scenarios, providing valuable insights into their effectiveness and efficiency.

Chapter 5

Simulations and Performance analysis

5.1 Introduction

As *IoT networks* become more complex, designing, implementing, and testing these systems, along with their underlying algorithms and protocols, has become increasingly challenging. To overcome these difficulties, researchers and developers frequently turn to simulation as a reliable and efficient approach. As a result, numerous simulation tools have been developed to support this process.

This chapter offers a thorough simulation-based analysis and performance evaluation of the proposed algorithms in various *IoT* network scenarios. Utilizing the *CupCarbon IoT* simulator, we systematically assess the effectiveness of these algorithms under varying cardinality constraints, network topologies, and operational conditions. To provide clarity and motivation, we begin by discussing *IoT* simulators, highlighting their importance, key challenges, and a review of the most widely used simulation tools. Next, we introduce *CupCarbon*, the simulator employed in this study, detailing its architecture and capabilities. Finally, we present our experimental setup, conduct a comparative analysis, and evaluate the proposed approach to ensure its efficiency and validity.

5.2 IoT simulators

This section underscores the significance of *IoT* simulators, outlines the principal challenges, and reviews the most commonly used simulation tools in the *IoT* domain.

5.2.1 Importance of IoT simulation

IoT simulators are crucial for the creation, evaluation, and investigation of *IoT* systems. They provide a controlled environment to evaluate performance, optimize protocols, and identify potential issues before real-world deployment. As *IoT networks* grow in complexity, simulation has become increasingly important for several reasons, the most well-known of which include cost-effective experimentation, scalability, prototyping and validation, and the ability to accelerate innovation while enhancing system reliability [136, 137].

- **Cost-Effective development:** installing equipment for testing can be expensive and operationally intricate. Simulators help mitigate these expenses by eliminating the need for purchasing, setting up, and maintaining hardware. Moreover, they allow developers to enhance resource efficiency by evaluating different situations and settings with no necessitating extra physical infrastructure.

- **Scalability testing:** *IoT networks* generally comprise numerous interconnected devices. Simulators allow developers to evaluate system performance against the strain of thousands or even millions of virtual devices without necessitating physical deployment. Moreover, simulating devices in different areas facilitates the analysis and optimization of data flow, latency, and resource management across various regions.

- **Prototyping and validation:** simulators allow developers to generate simulated models of IoT systems, facilitating validation and testing of functionalities prior to physical implementation. They also offer a regulated environment to assess diverse use cases and scenarios, guaranteeing the viability and efficacy of *IoT* solutions before implementation.

- **Risk mitigation:** simulators allow designers to induce faults and evaluate system answers, thereby ensuring the creation of resilient *IoT* systems without compromising actual devices or data. Moreover, they enable the simulation of cyber-attacks and security breaches, assisting in the identification of risks and the fortification of security measures with no subjecting the network to real threats.

5.2.2 Challenges in IoT simulation

A number of issues arise when simulating IoT systems, which must be carefully considered to ensure that the simulations are reliable and capable of managing large-scale operations. The most important ones include [137, 138]:

- **Security challenges:** *IoT* systems require the transmission of confidential information among multiple devices and sensors, making them vulnerable to cyber-attacks. Effective simulations must account for these security threats by accurately modeling potential attack scenarios. Understanding various attack vectors and replicating them in simulations is essential for developing robust security measures.

- **Heterogeneity challenges:** *IoT* ecosystems consist of a wide range of devices, communication protocols, and data formats, making simulation complex. Simulators must accurately replicate this diversity to ensure realistic testing. Handling different hardware architectures, operating systems, and network protocols is crucial for evaluating system performance and interoperability in real-world scenarios.

- **Energy consumption challenges:** *IoT* devices frequently operate with constrained battery life, rendering energy efficiency a pivotal consideration in system design. Simulations must precisely represent power consumption across diverse devices, communication protocols, and workloads to enhance energy efficiency. Implementing realistic energy constraints in simulations supports developers in creating more efficient *IoT* systems and prolonging device longevity.

- **Scalability challenges:** *IoT* environments includes an increasing array of interconnected devices that perpetually generate, process, and transmit data. With the escalation in scale and complexity of these interactions, the simulation of extensive *IoT networks* becomes progressively more difficult. Traditional simulation techniques and approaches, which are intended for smaller and less dynamic networks, frequently fail to accurately represent the complex behaviors and interactions of large-scale *IoT* systems.

5.2.3 Existing IoT simulators

IoT simulators emulate the functionality of connected devices, networks, and services to evaluate efficiency, capacity, and reliability. Essential attributes encompass device modeling, network communication, and data generation and processing. These simulators are crucial for evaluating *IoT* protocols, algorithms, inter-node communication, and network scalability. Here, we present several examples of *IoT* simulators [137, 139, 140].

- **Network Simulator (NS-2):** NS-2 is a widely used discrete event simulation tool originally developed as a general network simulator. Over time, it has expanded to support WSNs and the *IoT*. Its success is attributed to its flexibility, open-source availability, and robust features that enable users to design, test, and evaluate

various network types. NS-2 operates under a free license, fostering the development of additional modules. Simulations in NS-2 are implemented using C/C++ and Otcl (object-oriented Tcl extension), providing a powerful environment for network research and experimentation.

- **Network Simulator (NS-3):** NS-3 is a discrete event network simulator developed to address key limitations identified in NS-2. It is not a direct upgrade of NS-2 but rather a separate simulation platform designed to overcome issues such as the rigid coupling between C++ and Otcl in NS-2. Unlike its predecessor, NS-3 exclusively utilizes C++ and also supports Python, enhancing flexibility and usability. Notably, legacy NS-2 scripts written in Otcl are not compatible with NS-3, though efforts are ongoing to enable their integration. NS-3 is distributed under the GNU GPLv2 free license and is compatible with multiple platforms, including i386, x86-64, Linux, macOS, FreeBSD, Solaris, and Windows. It features a command-line configuration interface, while graphical support is available for certain animation functionalities through Nam.

- **OMNeT++:** OMNeT++ is a publicly available simulation platform developed in C++, designed to support various network components within an open simulation environment. It features an integrated development environment (IDE) with an embedded simulation kernel, enhancing usability and extensibility. Primarily intended for simulating communication networks, OMNeT++ is widely utilized in academic research due to its flexibility and modular architecture. Although originally developed for communication protocol simulations, its adaptability makes it well-suited for realistic simulations of IoT environments and applications.

- **Cooja:** Cooja is a flexible network simulator specifically designed for simulating IoT and wireless sensor networks (WSNs). It is an integral part of the Contiki OS, an open-source operating system optimized for resource-constrained IoT devices. Cooja enables simulation at multiple levels, including hardware emulation, network behavior analysis, and high-level application testing. It supports heterogeneous network topologies, allowing researchers and developers to test IoT protocols, algorithms, and large-scale deployments in a controlled environment. Due to its versatility and extensibility, Cooja is widely used for evaluating real-world IoT applications before deployment.

- **SimIoT:** SimIoT is a Java simulation toolkit designed for analyzing cloud computing systems within the IoT ecosystem. Built as an extension of SimIC, it incorporates user-generated data from IoT devices such as sensors and smartphones. The

toolkit models key entities and interactions, including users, data centers, and virtual machines, with a strong emphasis on real-time constraints. It features optimized message exchange and supports heterogeneous environments, making it adaptable to various cloud configurations. While SimIoT is particularly useful for applications in healthcare and information processing, it lacks real-world *IoT* implementation, physical device simulation, mobility support, and explicit energy efficiency mechanisms.

- **CupCarbon:** *CupCarbon* is a multi-agent discrete event simulation tool designed for researchers and developers to evaluate fundamental concepts of wireless sensor networks within a smart city environment. This simulator enables the assessment of various sensor types and facilitates the testing of wireless communication protocols across diverse network topologies. *CupCarbon* is particularly valued for its ability to analyze and optimize wireless topologies, assess protocol performance, and support scalability. Given the anticipated large-scale deployment of *IoT networks* with numerous interconnected nodes, a robust simulation platform like *CupCarbon* is essential for capturing the complex interactions and behaviors of such systems.

The choice of a simulator is a critical factor in determining the success of *IoT* research, emphasizing the importance of assessing the specific requirements of a study before selecting an appropriate simulation tool. Table 5.1 provides a summary of the aforementioned simulation tools, evaluated based on the established assessment criteria.

Table 5.1: Overview of Prominent Simulation Tools for IoT Networks

Features	**NS-2**	**NS-3**	**OMNeT++**	**Cooja**	**CupCarbon**
License	GPL	Open	Open	3-clause BSD	Open
Platform	Universal	Universal	TinyOS	Universal	Universal
Code	C++,C	C++, Pyton	C++,C	Java, C	Java
Scalability	Yes	Yes	Yes	Yes	Yes
Mobility	Yes	Yes	Yes	Yes	Yes
Protocol design /Optimization	Yes	Yes	Yes	Yes	Yes

5.3 CupCrbon IoT simulator

This section introduces the *CupCarbon IoT network* simulator, the primary tool examined in this thesis. It provides an overview of the key characteristics and functionalities of the simulator, its architectural framework, and examples of graph parameters implemented within *CupCarbon*.

5.3.1 CupCarbon plateform

To evaluate and validate our algorithms, we employed the widely recognized *IoT network* simulator, *CupCarbon* [1]. This simulator, accessible at [141], is highly regarded by researchers, developers, and academics [139, 142]. Its primary objective is to facilitate the design, visualization, debugging, and validation of *distributed algorithms*, particularly for tasks like monitoring, environmental data collection, and the creation of complex environmental scenarios [142]. The initial version of *CupCarbon* was introduced by Mehdi et al. [143], and since then, it has gained significant traction due to its versatility and utility. As a result, it has been widely adopted in various research domains.

CupCarbon is a discrete-event, multi-agent simulation tool designed for geolocation-based wireless sensor networks. It enables users to model and simulate networks on a digitized OpenStreetMap geographic interface. To facilitate realistic network simulations, *CupCarbon* provides a set of easily configurable and parameterizable objects. Figure 5.1 illustrates the graphical interface of this simulator.

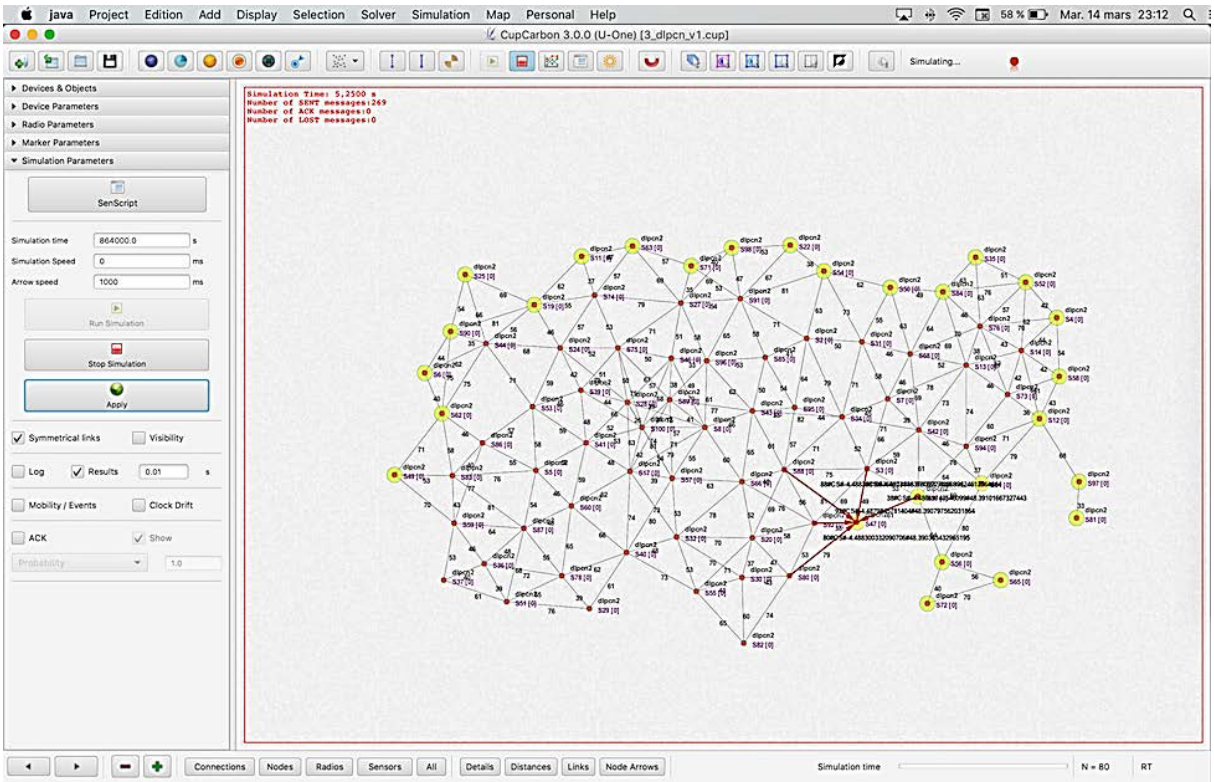


Figure 5.1: Graphical Interface of *CupCarbon*.

5.3.2 Architecture of CupCarbon

The *CupCarbon* simulator is designed with a modular architecture, allowing for easy customization and replacement of specific components. Figure 5.2 illustrates the platform's core modules, which are described as follows:

- **2D/3D City model module:** the main component of the simulator that is initially shown to the user is the 2D/3D City model module, which represents a digital format of a city and includes various information about the buildings, roads, locations, etc. It enables the deployment of the various sensor nodes of the network and is used to compute interferences and signal propagations.
- **Mobility module:** It makes it possible to design the mobiles' routes. A mobile device can be a sensor node, a device with a sensor node, or just a device without a communication system. The mobile can follow a predetermined trajectory if the mobility is fixed beforehand. In order to perform intelligent mobility, it can be decided in the script based on a specific scenario (such as the detection of a target, an abnormal sensed value, etc.).
- **Network module:** this module enables the design of the simulated wireless sensor network.
- **Communication Script module:** the *SenScript* language interpreter used to program each IoT node is represented by this module. The simulator will carry out each node's script's instructions.
- **Radio channel propagation module:** utilized to determine a group of network nodes' channel attenuation and impulse response. The resulting data is used to assess the wireless channel quality between node groups and determine whether or not a communication link can be established.
- **Interference module:** this module is used to determine whether or not the recipient will receive each message that is sent. It makes use of the previously mentioned models.
- **Simulation module:** the architecture's core is this module. Its foundation is the simulation of discrete events. An actual event, like mobility, or a natural event, like temperature, gas, etc., can also generate the events, as can the execution of each sensor node's script.

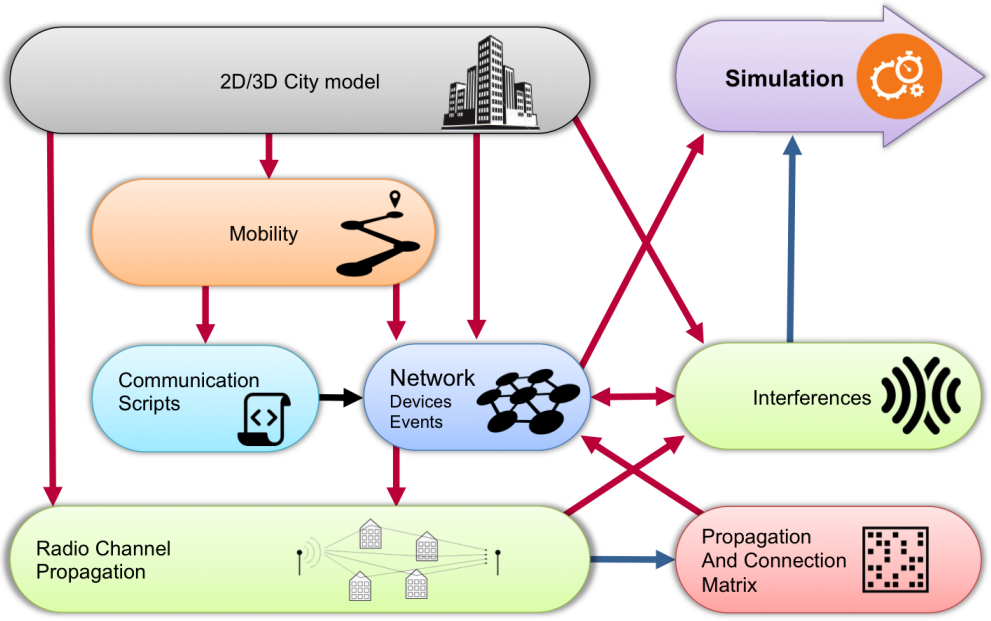


Figure 5.2: *CupCarbon* platform architecture [1].

5.3.3 CupCarbon and graph parameters

CupCarbon is a Smart City and *IoT* Wireless Sensor Network (WSN) simulator used for designing and testing sensor networks, including Zigbee, LoRa, and Wi-Fi networks. It provides an environment for simulating real-world sensor deployments and communication. It enables the analysis and optimization of wireless sensor networks by supporting key graph-theoretic concepts such as leader election, spanning tree construction, and domination analysis. These features are crucial for enhancing network efficiency, energy consumption, and communication reliability.

- **Shortest path:** *CupCarbon* provides tools to compute and analyze shortest path algorithms for optimizing communication in wireless sensor networks. The shortest path helps in efficient data routing, reducing energy consumption, and improving network performance. In [144], the authors examine the computation of the shortest path from the source node to the destination node in a wireless sensor network using the Dijkstra algorithm. They also compare various alternative algorithms, assessing their effectiveness and applicability across different application scenarios. In [145], the authors introduce a modified version of Dijkstra's algorithm that, in addition to considering the route cost, incorporates the remaining energy of nodes as an additional parameter to determine the optimal path.

- **Spanning tree:** a *spanning tree* is a subgraph that connects all nodes in the network without forming cycles, ensuring minimal redundancy and efficient communication.

It is used to optimize network connectivity, minimize energy consumption, and establish efficient routing structures in WSNs and *IoT* applications. In [72], the authors propose a method based on the flooding protocol to construct a spanning tree within a network. To identify the leaf nodes of the generated *spanning tree*, they employ the Flooding for Leaf Finding (FLF) Algorithm.

- **Domination:** the *dominating set problem* is fundamental in *IoT network* design, helping to achieve scalability, energy efficiency, and reliability. Many heuristic, metaheuristic, and machine learning approaches are now being explored to optimize dominating set. For instance, the authors of [71] proposed a distributed algorithm based on the WBS approach for identifying a *dominating set* in wireless sensor and *IoT networks*, optimizing selection in large-scale *IoT* deployments. Similarly, the authors of [146] introduced a novel *distributed algorithm* for identifying *dominating sets* in *IoT networks* under various criteria.

- **Leader election:** leader election is a fundamental problem in distributed systems and *IoT networks*, where nodes must collectively select a leader to coordinate tasks such as data aggregation, routing, energy management, and fault tolerance. In an *IoT network*, a leader can be the cluster head, master node, Decision-Making node, Blockchain validator, etc. Several distributed algorithms have been proposed for leader election in *IoT networks* using the *CupCarbon* simulator (see, for example, [45, 72–74] and the references therein).

In this thesis, we utilize the *CupCarbon* simulator to develop and implement *distributed algorithms* of our approach, focusing on *secure dominating set*, *independent set*, and *critical node set* [147].

5.4 Experimental results and Comparaison

In this section, we present our experimental results and the necessary comparisons. For each graph parameter considered in our approach, the experiment is executed in two phases. The initial phase involves designing the *IoT network* utilizing *CupCarbon's* user-friendly graphical interface, which incorporates OpenStreetMap for precise geospatial representation, facilitating accurate positioning of *IoT* nodes. During the second phase, each node is configured utilizing *SenScript*, a domain-specific scripting language created for *CupCarbon*. Then, we perform a comparative analysis using standard performance metrics to assess the efficacy of our approach.

5.4.1 Secure dominating set problem

5.4.1.1 Experimental results

We designed and deployed an *IoT network* consisting of 200 interconnected nodes. To efficiently manage the network, we adapted and implemented the algorithm proposed by Bounsseur et al. [72] to identify the *spanning tree* within this network. Furthermore, we developed and integrated our proprietary *SDSTree* algorithm using the *SenScript* language to accurately determine the secure dominating nodes. As shown in Figure 5.3 the successful execution of the *SDSTree* algorithm on the spanning tree demonstrates its effectiveness in enhancing the network's security and structural integrity.

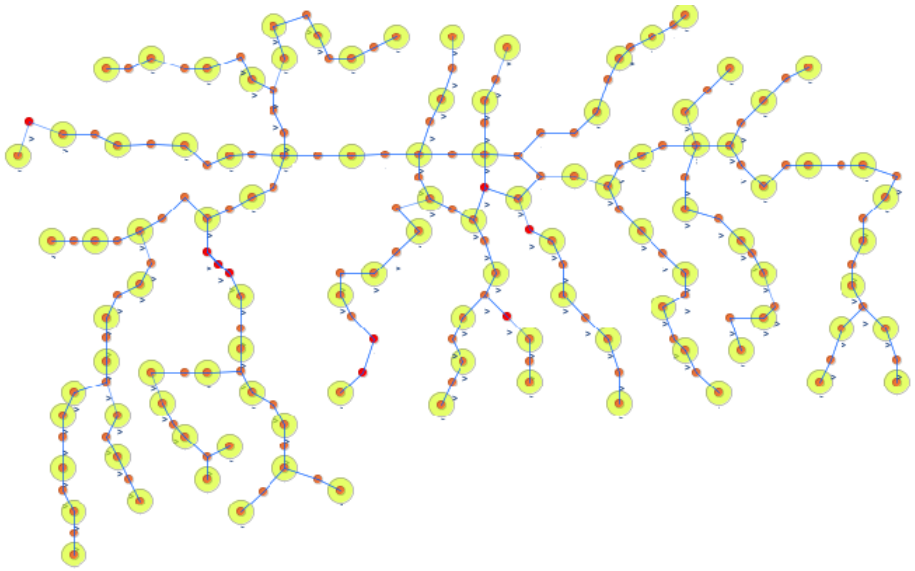


Figure 5.3: An illustration of 98 nodes that secure dominate 200 nodes.

5.4.1.2 Comparison

We perform a thorough comparison between the *WBS-MDS* algorithm, which finds a *minimal dominating set* using common comparison metrics, and the *SDSTree* algorithm, which is specifically made to find a *secure dominating set* in *IoT networks* [71]. Our experiments are carried out only on networks with a tree topology because the *WBS-MDS* algorithm is made for *IoT networks* with arbitrary topologies, and the crucial part of our method (step 2) is running the *SDSTree* algorithm on a tree-structured network. To ensure the accuracy and reliability of our findings, we systematically vary the network size and replicate the experiments under controlled conditions. Table 5.2 presents a detailed analysis of the results for each network configuration, including the number of selected

nodes, average execution time (in seconds), and the total count of sent and received (SR)-messages for both algorithms.

Table 5.2: Comparative Analysis of *SDSTree* and *WBS-MDS* algorithms on random networks, regarding Node Selection, CPU Execution Time, and Communication Overhead (Sent/Received Messages).

	SDSTree algorithm			WBS-MDS algorithm		
N	SDS	execution time	(SR)-messages	DS	execution time	(SR)-messages
50	28	45.638	142	21	93.805	142
100	69	76.032	262	42	154.613	285
150	70	111.275	411	60	346.225	430
200	95	66.529	525	78	400.635	583
250	124	76.351	675	101	528.857	733
300	162	71.452	841	129	616.803	858
350	172	76.703	900	151	676.234	993
400	202	96.971	1049	170	2699.769	1135
450	216	92.249	1131	187	765.311	1278
500	251	98.528	1259	206	772.523	1429

- **Performance comparison:** the primary comparison metric evaluates the number of *dominating nodes* and *secure dominating nodes* identified by applying both algorithms to each input network. The results are presented in Figure 5.4.

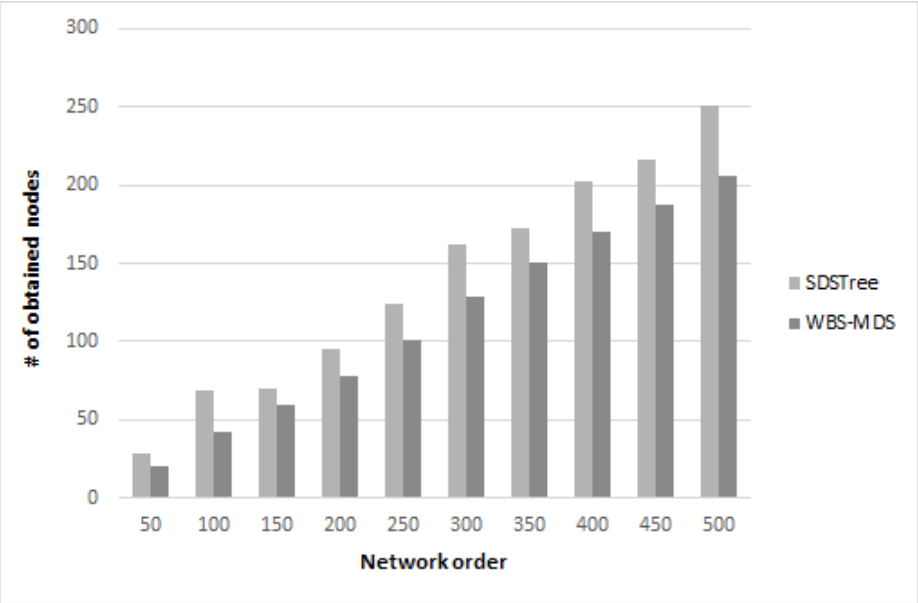


Figure 5.4: Comparison of the nodes acquired between the *SDSTree* and *WBS-MDS* algorithms.

A direct analysis of the results presented in histogram 5.4 reveals that, for every value of n across all input networks, the *SDSTree* algorithm consistently yields a higher number of secure dominating nodes compared to the number of dominating nodes identified by the *WBS-MDS* algorithm. This result is a direct consequence of the inequality $\gamma_s(G) \geqslant \gamma(G)$, as established in Theorem 2.1. Furthermore, it is noteworthy that the number of secure dominating nodes identified by our algorithm closely approximates $j = 3n$ for the path tree, as demonstrated in Theorem 2.2.

- **Convergence comparison :** the second comparison metric assesses the execution time of each algorithm across the input networks. The corresponding results are presented in Figure 5.5.

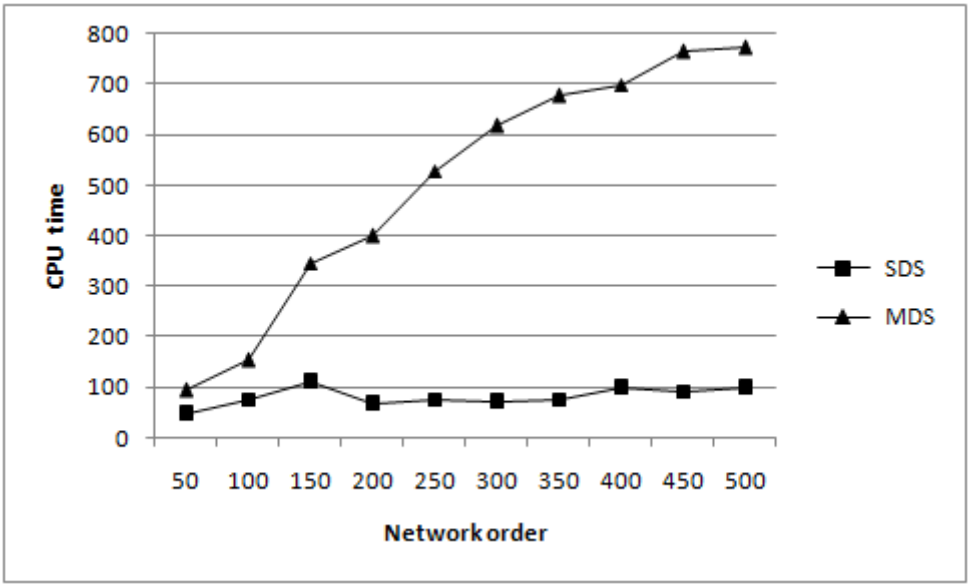


Figure 5.5: A comparison of the *SDSTree* and *WBS-MDS* algorithms' execution times.

These results demonstrate that the *SDSTree* algorithm consistently outperforms the *WBS-MDS* algorithm in terms of execution time across all input networks. The delay observed in the *WBS-MDS* algorithm is primarily due to the idling period required before *dominating nodes* initiate execution. In contrast, the *SDSTree* algorithm enables nodes with a degree of 1 to commence execution immediately, thereby eliminating unnecessary delays. Moreover, the execution speed of the *SDSTree* algorithm remains stable regardless of network size, further underscoring its efficiency.

It is important to note that minor variations or perturbations may appear in the diagram due to factors related to the execution environment, such as memory and processor specifications. Additionally, our complexity analysis of the *WBS-MDS* algorithm determined its complexity to be $(\Delta+1)$, which exceeds that of the *SDSTree*

algorithm. This difference in complexity further highlights the superior execution efficiency of our proposed approach.

- **Communication comparison:** the third comparison metric evaluates the total number of sent and received *(SR)-messages*. To analyze this, we executed both proposed algorithms on networks of varying sizes and generated the corresponding histograms presented in Figure. 5.6.

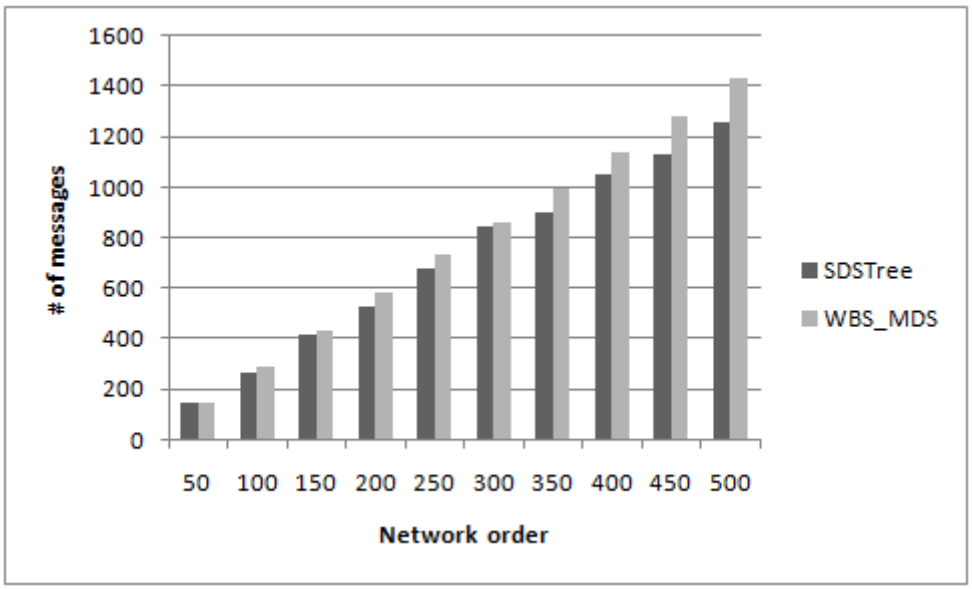


Figure 5.6: Comparison of the aggregate quantity of transmitted and received messages as determined by the *SDSTree* and *WBS-MDS* algorithms.

This histogram demonstrates that the *SDSTree* algorithm consistently requires fewer message exchanges compared to the *WBS-MDS* algorithm across all experimental network configurations. The higher message overhead in the *WBS-MDS* algorithm can be attributed to the increased number of iterations required for processing messages and updating neighbor states.

5.4.2 Maximal independent set problem

5.4.2.1 Experimental results

In our experimentation, we used the value 100 for both the variables *wut* and *ub* as input. Figure 5.7 illustrates an example of an *IoT network* with 200 nodes randomly generated in *CupCarbon* simulator, where the node degrees are between 1 and 10. The yellow nodes represent the *maximal independent set*.

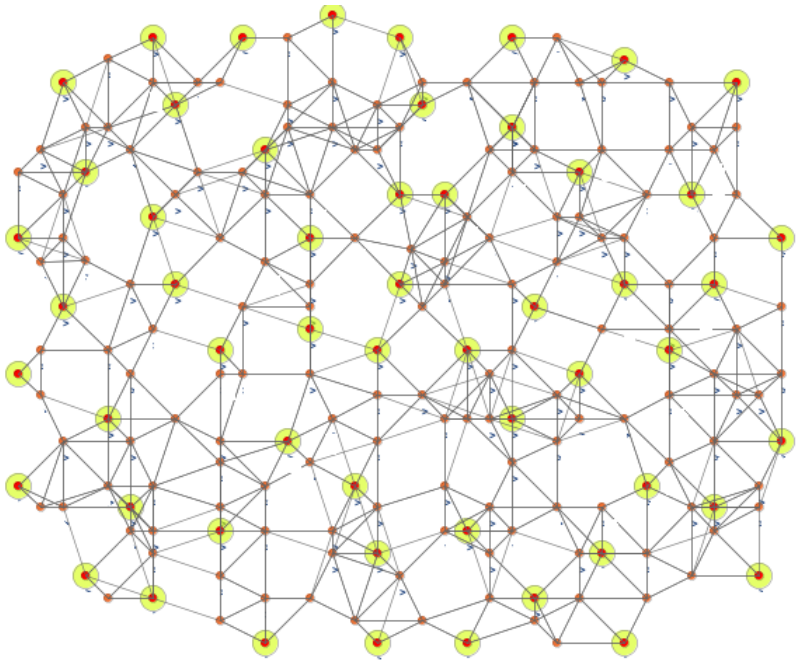


Figure 5.7: Example of 55 independent nodes over 200 nodes.

5.4.2.2 Comparison

Since our *WBS-MIS* algorithm is the first *WBS-based* solution developed to address the *MISP*, we will compare its performance with an interesting *WBS-based* algorithm [71], called *WBS-MDIS*, designed for identifying a *minimal dominating independent set*. Although these algorithms solve distinct problems, they share common characteristics, such as their reliance on distributed computation and node selection. To ensure a meaningful comparison, we will focus on metrics that reflect these shared properties, including execution time and the number of messages exchanged. This will allow us to evaluate both algorithms in terms of efficiency and communication overhead, offering insights into their respective strengths and suitability for different *IoT network* applications. To do this comparison, we gradually increase the input network's order and replicate the experiment. Table 5.3 presents, for each network and both algorithms, the count of nodes obtained, the average running times (in seconds), and the total number of sent and received (SR)-messages.

- **Convergence comparison:** the first comparison metric is the execution time of each algorithm on each input network. The obtained results are shown in Figure 5.8.

Table 5.3: Experimental results of *WBS-MIS* and *WBS-MDIS* on different random networks regarding the number of obtained nodes, CPU time, and sent/received messages.

	WBS-MIS algorithm			WBS-MDIS algorithm		
N	MIS	execution time	(SR)-messages	MIDS	execution time	(SR)-messages
100	29	38.515	614	23	45.17	614
200	53	68.294	1366	43	76.17	1366
300	84	100.386	1774	64	98.829	1774
400	112	136.363	2406	83	119.497	2406
500	134	177.297	3512	103	172.228	3512
600	159	184.99	4238	128	201.865	4194
700	180	206.76	5031	146	198.171	5001
800	217	256.907	5631	172	245.457	5624
900	244	256.017	6316	182	271.12	6292
1000	270	356.44	6954	214	308.609	6953

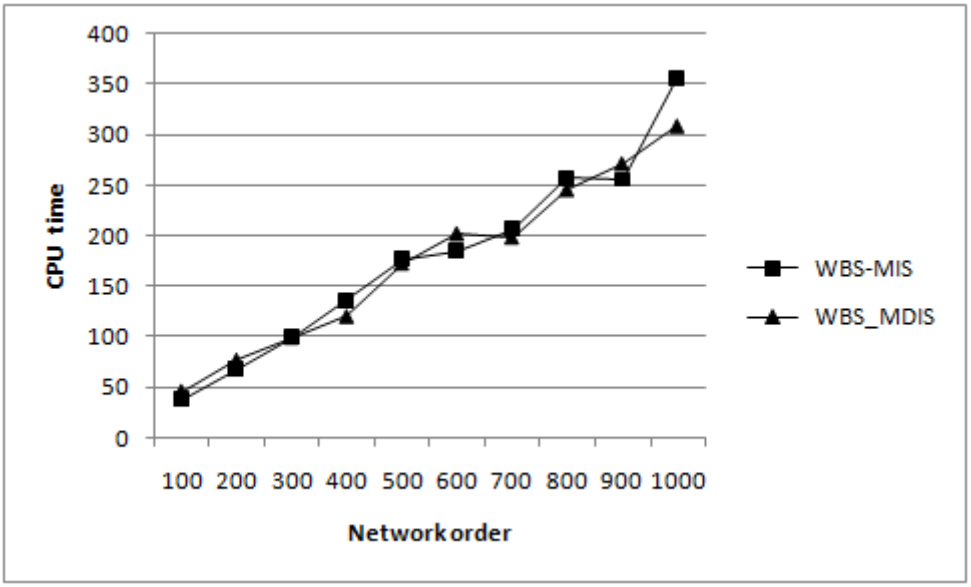


Figure 5.8: Comparison of the *execution time* between the *WBS-MIS* and *WBS-MDIS* algorithms.

As illustrated this figure, both algorithms exhibit similar trends in execution time as the network size increases. The *WBS-MIS* algorithm consistently demonstrates slightly faster convergence across all network sizes. For example, in a network with 300 nodes, the *WBS-MIS* algorithm converges in 100.386 seconds, while *WBS-MDIS* requires 98.829 seconds. The difference remains small, with the *WBS-MIS* algorithm maintaining a competitive execution time, even as the network size grows to 1000 nodes, where the algorithm completes in 356.44 seconds compared to 308.609

seconds for *WBS-MDIS*. This marginal difference can be attributed to the similar complexity of both algorithms. As noted in the complexity analysis, both *WBS-MIS* and *WBS-MDIS* have a time complexity of $O(\Delta + 1)$, where Δ is the maximum degree of the network. This suggests that the number of iterations required for convergence in both algorithms scales linearly with the maximum node degree, resulting in comparable performance as network sizes increase.

- **Communication comparison:** the second comparison metric is the total number of sent and received (SR)-messages. The communication comparison between the *WBS-MIS* and *WBS-MDIS* algorithms is essential to understanding the trade-offs in terms of message overhead in distributed *IoT network* environments. Communication efficiency is particularly crucial in resource-constrained *IoT networks*, where minimizing message exchanges directly translates into energy savings and network longevity.

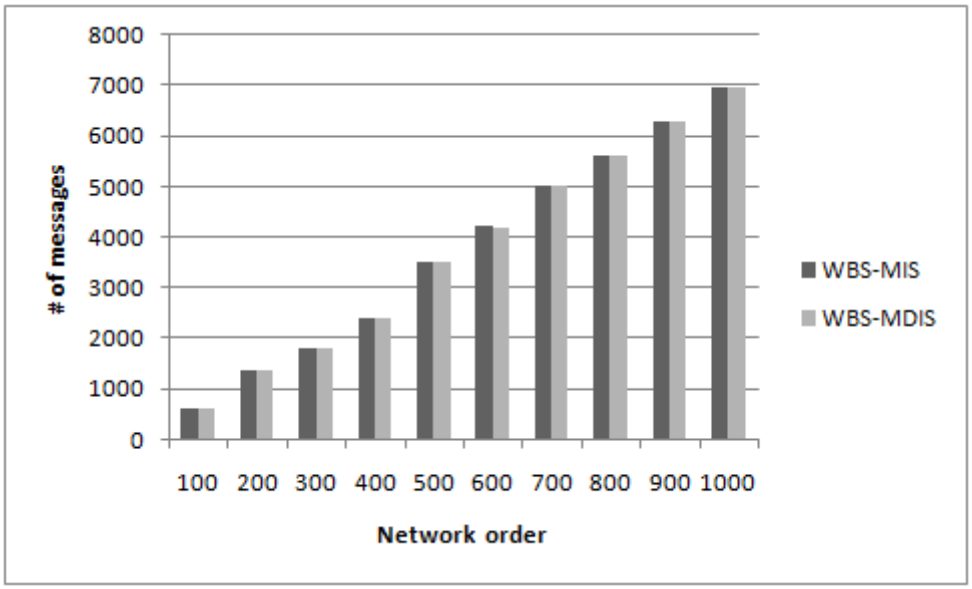


Figure 5.9: Comparison of the aggregate quantity of transmitted and received messages as determined by the *WBS-MIS* and *WBS-MDIS* algorithms.

As shown in Figure 5.9, both algorithms consistently exchange the same number of sent and received messages (SR-messages) across different network sizes, highlighting their efficiency. For example, 614 messages are exchanged in 100-node networks, scaling to 6,954 messages in 1,000-node networks. Communication remains uniform in both *WBS-MIS* and *WBS-MDIS*, as nodes broadcast their status based on their calculated waiting time, ensuring synchronization with the network structure and minimizing redundant exchanges.

5.4.3 Critical node detection problem

5.4.3.1 Experiment

This subsubsection demonstrates the applicability of the *DA3C-CNP* distributed algorithm. In order to accomplish this, we take the identical example presented (refer to [148]). This representative example was initially proposed in [115] by the authors who introduced this variant graph parameter. The experiment begins with the design of an *IoT network* using a tree topology, created through *CupCarbon's* intuitive graphical interface. Then, each *IoT* node is configured using *SenScript*. *Critical nodes*, delineated by the *DA3C-CNP* algorithm, are marked in yellow, as illustrated in Figure 5.10.

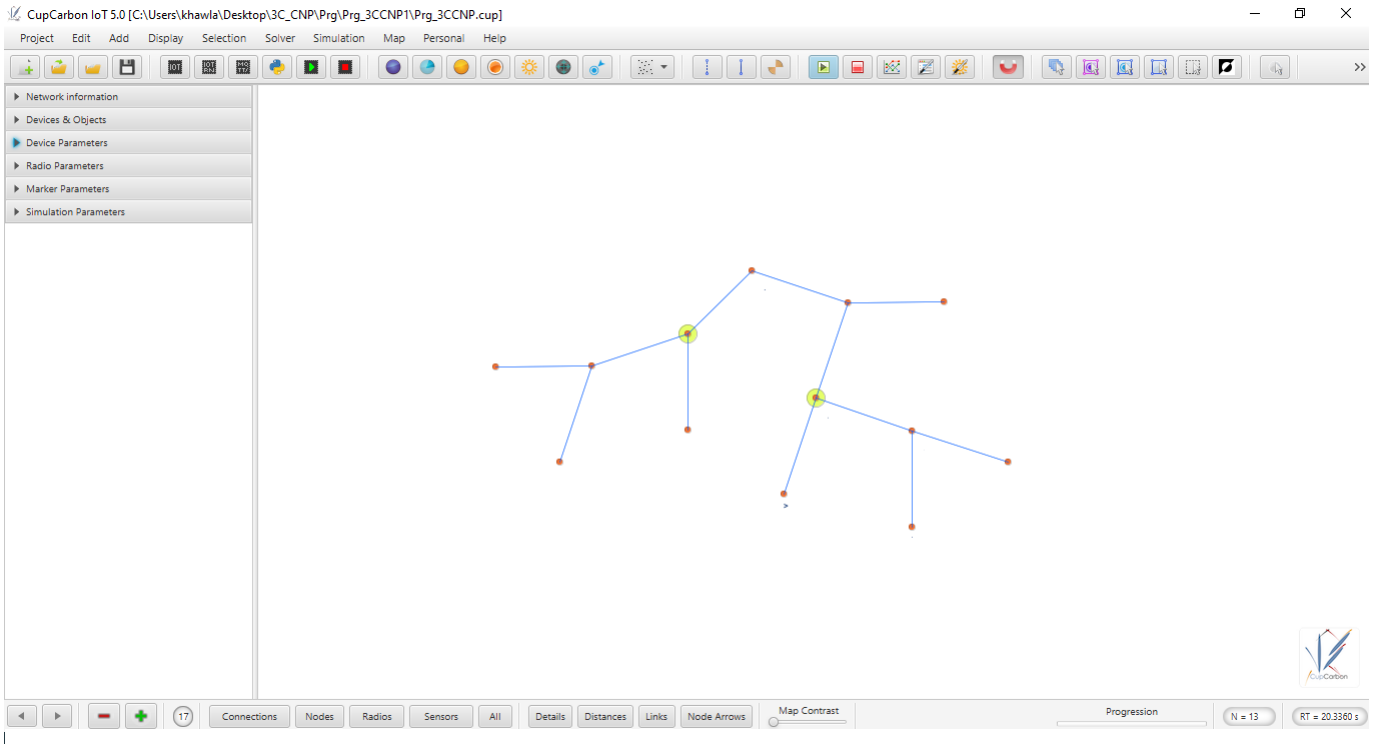


Figure 5.10: Illustration of 2 *critical nodes* among 13 nodes utilizing the *CupCarbon IoT* simulator.

5.4.3.2 Comparison

The *DA3C-CNP* algorithm presents considerable benefits regarding time complexity when juxtaposed with current centralized solutions for the 3C-CNP problem. The authors of [115] attained a time complexity of $O(n^2)$ for proper interval graphs and weighted trees, while our *DA3C-CNP* algorithm achieves a linear time complexity of $O(n - l)$, rendering it more efficient for large-scale environments. In a similar vein, although [115]

attained $O(n)$ for unweighted trees, our distributed algorithm improves scalability and parallelization, rendering it especially appropriate for distributed settings.

Moreover, in contrast to the dynamic programming algorithm presented in [124] for bipartite permutation graphs, which exhibits a complexity of $O(nB^2)$, our algorithm removes the reliance on the parameter B, resulting in enhanced consistency and predictability in performance. Furthermore, whereas the recent study in [135] concentrates on chordal graphs with a maximum node degree of $\Delta = 3$, the $O(n - l)$ complexity of our algorithm is unaffected by node degree limitations, thereby rendering it suitable for a broader array of situations. The *DA3C-CNP* algorithm exhibits enhanced efficiency and scalability, successfully overcoming the disadvantages of current methods.

5.5 Conclusion

In this chapter, we presented the simulation results and performance analysis of the proposed algorithms across various *IoT network* scenarios. Using the *CupCarbon IoT* simulator, we systematically evaluated each algorithm's effectiveness under different cardinality constraints and network conditions. These simulations provided valuable insights into the algorithms' scalability, efficiency, and suitability for real-world *IoT* applications, with a focus on their ability to address specific network challenges. The analysis began with the *secure dominating set problem*, followed by the *maximal independent set problem*, and concluded with the *critical node set problem*, allowing for a comprehensive evaluation of each algorithm's strengths and limitations across diverse *IoT* contexts.

General conclusion

This thesis aims to enhance the security mechanisms of *IoT networks* using a graph-based approach. Given the inherently distributed nature of *IoT* environments, we propose decentralized algorithmic solutions to determine critical security parameters. Our approach integrates theoretical foundations with practical implementations and offers three principal contributions centered on secure graph parameter solutions tailored for distributed IoT networks:

- First contribution: we investigated the *SDSP* as a security parameter and proposed a distributed solution to address it within IoT networks. The proposed solution operates in two phases: the first phase involves the *spanning tree* construction, which establishes a hierarchical network structure, and the second phase focuses on calculating the *secure dominating set*, where *secure dominating nodes* are identified based on their position and connectivity within the tree (e.g., leaf and support nodes). For the first phase, we utilized an existing algorithm from the literature, and in the second phase, we introduced our distributed algorithm, *SDSTree*. The *SDSTree* algorithm converges within $\Delta - 1$ iterations per node and requires the exchange of n messages.

- Second contribution: we explored the *MISP* in graphs and networks, and developed a *distributed algorithm* called *WBS-MIS*, utilizing the *WBS* technique. This algorithm allows for efficient computation of the *maximal independent set* in large-scale networks, improving both scalability and performance in distributed environments. The *WBS-MIS* algorithm assigns a calculated weighted value to each node to determine the precise timing for its execution. Complexity analyses demonstrated that the *WBS-MIS* algorithm converges within $\Delta + 1$ iterations. Additionally, the algorithm requires the exchange of n messages, which is optimal for distributed *IoT* applications.

- Third contribution: we investigated the *critical node* parameter to identify a minimal subset of nodes whose removal most significantly impacts network connectivity.

To tackle this, we suggested and tested the *DA3C-CNP* algorithm, which is a distributed method aimed at solving the *3C-CNP* variant in tree-based *IoT networks*. The *DA3C-CNP* algorithm converges within $\Delta - 1$ iterations per node and requires the exchange of n messages.

To validate the proposed approach, we used the *Cupcarbon IoT* simulator, which is a widely recognized tool for designing *IoT networks* and validating *distributed algorithms*. While the proposed algorithms in our approach follow a greedy method and do not guarantee an optimal solution, they deliver competitive performance, converging in linear rounds with the same order of round complexity as the most efficient algorithms in the literature. This makes it particularly suitable for real-time applications where rapid decision-making is essential. While the algorithms proposed in our approach have proven effective in addressing specific security challenges within *IoT networks*, several limitations remain, providing avenues for future research:

- Optimizing heuristic solutions: although the heuristic nature of the algorithms offers significant performance improvements, they do not guarantee optimal solutions. Future research could explore more advanced heuristic or metaheuristic algorithms that deliver solutions closer to optimal, while maintaining efficiency, especially for critical *IoT* applications that require high reliability.

- Scalability for large-scale networks: the current evaluations of the proposed algorithms, such as *DA3C-CNP*, have primarily focused on moderately sized networks. Future research should aim to enhance the scalability of these methods to accommodate larger *IoT networks* containing thousands, or even millions, of nodes. This would require optimizing the algorithms to maintain low time complexity and communication overhead, even as the network size grows.

- Expanding the *WBS* approach: the *WBS* technique, which has proven effective in the context of the *MISP* and leader election, shows potential for application to other graph parameters, such as vertex cover, matching, vertex coloring, critical node detection, etc. Future research could explore these extensions, particularly investigating the feasibility of using *WBS-MIS* to solve the *CNDP* in *IoT networks*. *CNDP*, which involves identifying nodes whose removal maximally disrupts connectivity, remains a critical focus in network security. Given its reliance on connectivity metrics, addressing *CNDP* could benefit from techniques related to *MISP*.

Bibliography

[1] Ahcène. Bounceur. Cupcarbon: a new platform for designing and simulating smart-city and iot wireless sensor networks (sci-wsn). In *Proceedings of the International Conference on Internet of things and Cloud Computing(ICC'2016)*, pages 1–1, 2016.

[2] Shoumen Palit Austin Datta. Auto id paradigm shifts from internet of things to unique identification of individual decisions in system of systems. *MIT ESD Working Paper Series and Supply Chain Europe*, 2008.

[3] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[4] Zhiqiang Gao, Luqman Ali, Cong Wang, Ruizhi Liu, Chunwei Wang, Cheng Qian, Hokun Sung, and Fanyi Meng. Real-time non-contact millimeter wave radar-based vital sign detection. *Sensors*, 22(19):7560, 2022.

[5] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[6] Hermann Kopetz and Wilfried Steiner. Internet of things. In *Real-time systems: design principles for distributed embedded applications*, pages 325–341. Springer, 2022.

[7] Yasmine Harbi, Zibouda Aliouat, Allaoua Refoufi, and Saad Harous. Recent security trends in internet of things: A comprehensive survey. *IEEE Access*, 9:113292–113314, 2021.

[8] Marco Lombardi, Francesco Pascale, and Domenico Santaniello. Internet of things: A general overview between architectures, protocols and applications. *Information*, 12(2):87, 2021.

[9] Mohamed Tahar Hammi. *Sécurisation de l'Internet des objets*. PhD thesis, Université Paris Saclay (COmUE), 2018.

[10] Maroua Ahmid and Okba Kazar. A comprehensive review of the internet of things security. *Journal of Applied Security Research*, 18(3):289–305, 2023.

[11] Mukrimah Nawir, Amiza Amir, Naimah Yaakob, and Ong Bi Lynn. Internet of things (iot): Taxonomy of security attacks. In *2016 3rd international conference on electronic design (ICED)*, pages 321–326. IEEE, 2016.

[12] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, and Parvez Faruki. Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys & Tutorials*, 21(3):2671–2701, 2019.

[13] Linghe Kong, Jinlin Tan, Junqin Huang, Guihai Chen, Shuaitian Wang, Xi Jin, Peng Zeng, Muhammad Khan, and Sajal K Das. Edge-computing-driven internet of things: A survey. *ACM Computing Surveys*, 55(8):1–41, 2022.

[14] Onur Ugurlu, Nusin Akram, and Vahid Khalilpour Akram. Critical nodes detection in iot-based cyber-physical systems: Applications, methods, and challenges. In *Emerging trends in IoT and integration with data science, cloud computing, and big data analytics*, pages 226–239. IGI Global, 2022.

[15] Indira Kalyan Dutta, Bhaskar Ghosh, and Magdy Bayoumi. Lightweight cryptography for internet of insecure things: A survey. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0475–0481. IEEE, 2019.

[16] Goiuri Peralta, Raul G Cid-Fuentes, Josu Bilbao, and Pedro M Crespo. Homomorphic encryption and network coding in iot architectures: Advantages and future challenges. *Electronics*, 8(8):827, 2019.

[17] Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. Machine learning in iot security: Current solutions and future challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1686–1721, 2020.

[18] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. *IEEE communications surveys & tutorials*, 22(3):1646–1685, 2020.

[19] KK Sreelakshmi, Ashutosh Bhatia, and Ankit Agrawal. Securing iot applications using blockchain: a survey. *URL https://arxiv. org/abs/2006.03317, accessed*, 10, 2020.

[20] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *Int. J. Netw. Secur.*, 19(5):653–659, 2017.

[21] Ali A Ghorbani, Wei Lu, and Mahbod Tavallaee. *Network intrusion detection and prevention: concepts and techniques*, volume 47. Springer Science & Business Media, 2009.

[22] Mohammed Lalou, Mohammed Amin Tahraoui, and Hamamache Kheddouci. The critical node detection problem in networks: A survey. *Computer Science Review*, 28:92–117, 2018.

[23] Irène Charon and Olivier Hudry. *Introduction à l'optimisation continue et discrète*. Lavoisier, 2019.

[24] Mourad GUETTICHE. Paramètre de graphes et protection des réseaux (doctoral dissertation),university of bejaia, 2019.

[25] HR Lewis and R Michael. Garey and david s. *Johnson."Computers and intractability. A guide to the theory of NP-completeness." WH Freeman and Company, San Francisco*, pages 498–500, 1979.

[26] Michael R Garey and David S. Johnson. Computers and intractability, vol. 29, 2002. wh freeman New York.

[27] Derek G Corneil and Lorna K Stewart. Dominating sets in perfect graphs. *Discrete Mathematics*, 86(1-3):145–164, 1990.

[28] Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[29] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Szegedy Mario. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[30] Johan Hstad. Clique is hard to approximate within n1. In *Proc. 37th Symp. on Found. Comput. Sci*, pages 627–636, 1996.

[31] Stefano. Basagni. Finding a maximal weighted independent set in wireless networks. *Telecommun Syst.*, 18:155–168, 2001.

[32] Sujay Sanghavi, Devavrat Shah, and Alan S Willsky. Message passing for maximum weight independent set. *IEEE Trans. Inf. Theory*, 55(11):4822–4834, 2009.

[33] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977.

[34] Endre Boros, Khaled M. Elbassioni, Vladimir A. Gurvich, and Leonid Khachiyan. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Process. Lett.*, 10(4):253–266, 2000.

[35] Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM J. Comput.*, 9(3):558–565, 1980.

[36] David Eppstein. All maximal independent sets and dynamic dominance for sparse graphs. *ACM Trans. Algorithms*, 5(4):1–14, 2009.

[37] Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca. Versari. Listing maximal independent sets with minimal space and bounded delay. In String Processing and Information Retrieval, editors, *24th International Symposium, SPIRE 2017, September 26–29, 2017, Proceedings 24*, volume 42, pages 144–160, Palermo, Italy, 2017. Springer.

[38] Hideyuki Tanaka, Yuichi Sudo, Hirotsugu Kakugawa, Toshimitsu Masuzawa, and Ajoy K. Datta. A self-stabilizing 1-maximal independent set algorithm. *J. Inf. Process.*, 29:247–255, 2021.

[39] Zhengnan Shi, Wayne Goddard, and Stephen T Hedetniemi. An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. *Inf. Process. Lett.*, 91(2):77–83, 2004.

[40] Ajay D Kshemkalyani and Mukesh Singhal. *Distributed computing: principles, algorithms, and systems.* Cambridge University Press, 2011.

[41] Kayhan Erciyes. *Distributed graph algorithms for computer networks.* Springer Science & Business Media, 2013.

[42] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

[43] K Erciyes. *Guide to graph algorithms.* Springer, 2018.

[44] Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang, Xuemin Lin, and Jingren Zhou. A survey of distributed graph algorithms on massive graphs. *ACM Computing Surveys*, 57(2):1–39, 2024.

[45] Ahcène Bounceur, Madani Bezoui, Reinhardt Euler, and Farid. Lalem. A wait-before-starting algorithm for fast, fault-tolerant and low energy leader election in wsns dedicated to smart-cities and iot. In *2017 IEEE SENSORS*, pages 1–3, Glasgow, UK, 2017. IEEE.

[46] Praveen Kumar Donta, Ilir Murturi, Victor Casamayor Pujol, Boris Sedlak, and Schahram Dustdar. Exploring the potential of distributed computing continuum systems. *Computers*, 12(10):198, 2023.

[47] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of grid computing*, 3:171–200, 2005.

[48] Jiangshui Hong, Thomas Dreibholz, Joseph Adam Schenkel, and Jiaxi Alessia Hu. An overview of multi-cloud computing. In *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*, pages 1055–1068. Springer, 2019.

[49] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 96–107, 2002.

[50] Rajkumar Buyya and Satish Narayana Srirama. *Fog and edge computing: principles and paradigms*. John Wiley & Sons, 2019.

[51] Nour Alhuda Sulieman, Lorenzo Ricciardi Celsi, Wei Li, Albert Zomaya, and Massimo Villari. Edge-oriented computing: A survey on research and use cases. *Energies*, 15(2):452, 2022.

[52] Houcine Boumediene Merouane and Mustapha Chellali. On secure domination in graphs. *Information Processing Letters.*, 115(10):786–790, 2015.

[53] Ernie J Cockayne, PJP Grobler, WR Grundlingh, J Munganga, and Jan H Van Vuuren. Protection of a graph. *Utilitas Mathematica*, 67:19–32, 2005.

[54] Ernest J Cockayne. Irredundance, secure domination and maximum degree in trees. *Discrete mathematics*, 307(1):12–17, 2007.

[55] EJ Cockayne, O Favaron, and CM Mynhardt. Secure domination, weak roman domination and forbidden subgraphs. *Bull. Inst. Combin. Appl*, 39:87–100, 2003.

[56] S Benecke, EJ Cockayne, and CM Mynhardt. Secure total domination in graphs. *Utilitas Mathematica*, 74:247–260, 2007.

[57] William Klostermeyer and Christina Mynhardt. Secure domination and secure total domination in graphs. *Discussiones Mathematicae Graph Theory.*, 28(2):267–284, 2008.

[58] CM Mynhardt, HC Swart, and E Ungerer. Excellent trees and secure domination. *Utilitas Mathematica.*, 67:255–267, 2005.

[59] Elmer C Castillano, RL Ugbinada, and Sergio R Canoy Jr. Secure domination in the join of graphs. *Applied Mathematical Sciences.*, 8(105):5203–5211, 2014.

[60] Zepeng Li and Jin Xu. A characterization of trees with equal independent domination and secure domination numbers. *Information Processing Letters.*, 119:14–18, 2017.

[61] Alewyn P Burger, Ernest J Cockayne, WR Grundlingh, Christina M Mynhardt, Jan H van Vuuren, and Wynand Winterbach. Infinite order domination in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing.*, 50:179–194, 2004.

[62] Wayne Goddard, Sandra M Hedetniemi, and Stephen T Hedetniemi. Eternal security in graphs. *J. Combin. Math. Combin. Comput.*, 52:169–180, 2005.

[63] Alewyn P Burger, Anton Pierre de Villiers, and Jan H van Vuuren. A linear algorithm for secure domination in trees. *Discrete Applied Mathematics.*, 171:15–27, 2014.

[64] Toru Araki and Hiroka Miyazaki. Secure domination in proper interval graphs. *Discrete Applied Mathematics.*, 247:70–76, 2018.

[65] AP Burger, AP De Villiers, and JH Van Vuuren. Two algorithms for secure graph domination. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 85:321–339, 2013.

[66] Xiuyang Chen, Changbing Tang, and Zhao Zhang. A game theoretic approach for a minimal secure dominating set. *IEEE/CAA Journal of Automatica Sinica*, 10(12):2258–2268, 2023.

[67] Tanguy Godquin, Morgan Barbier, Chrystel Gaber, Jean-Luc Grimault, and Jean-Marie Le Bars. Applied graph theory to security: A qualitative placement of security solutions within iot networks. *Journal of information security and applications*, 55:102640, 2020.

[68] Djamila Bendouda, Abderrezak Rachedi, and Hafid Haffaf. Programmable architecture based on software defined network for internet of things: connected dominated sets approach. *Future Generation Computer Systems*, 80:188–197, 2018.

[69] V Ceronmani Sharmila and A George. Implementation of connected dominating set in fog computing using knowledge-upgraded iot devices. *Knowledge Computing and Its Applications: Knowledge Manipulation and Processing Techniques*, 1:3–24, 2018.

[70] Zulfiqar Ali Zardari, Jingsha He, Nafei Zhu, Khalid Hussain Mohammadani, Muhammad Salman Pathan, Muhammad Iftikhar Hussain, and Muhammad Qasim Memon. A dual attack detection technique to identify black and gray hole attacks using an intrusion detection system and a connected dominating set in manets. *Future Internet*, 11(3):61, 2019.

[71] Madani Bezoui, Ahcene Bounceur, Reinhardt Euler, Farid Lalem, and Laouid Abdelkader. A new algorithm for finding a dominating set in wireless sensor and iot networks based on the wait-before-starting concept. In *2017 IEEE SENSORS*, pages 1–3. IEEE, 2017.

[72] Ahcène Bounceur, Madani Bezoui, Reinhardt Euler, Nabil Kadjouh, and Farid Lalem. Brogo: a new low energy consumption algorithm for leader election in wsns. In *2017 10Th international conference on developments in esystems engineering (deSE)*, pages 218–223, 2017.

[73] Ahcene Bounceur, Madani Bezoui, Massinissa Lounis, Reinhardt Euler, and Ciprian Teodorov. A new dominating tree routing algorithm for efficient leader election in iot networks. In *CCNC 2018 - 2018 15th IEEE Annual Consumer Communications and Networking Conference*, CCNC 2018 - 2018 15th IEEE Annual Consumer Communications and Networking Conference, pages 1–2, United States, March 2018. Institute of Electrical and Electronics Engineers Inc.

[74] Nabil Kadjouh, Ahcène Bounceur, Madani Bezoui, Mohamed Essaid Khanouche, Reinhardt Euler, Mohammad Hammoudeh, Loïc Lagadec, Sohail Jabbar, and Fadi Al-Turjman. A dominating tree based leader election algorithm for smart cities iot infrastructure. *Mobile Networks and Applications*, 28:1–14, 2023.

[75] Andrew S Tanenbaum and David J Wetherall. Computer networks, 5th. pearson education, inc. ISBN 978-0-13-212695-3, 2011.

[76] Leslie G Valiant. Parallel computation. In *Proc. 7th IBM Symp. on Math. Found.of Comp. Sci.*, 1982.

[77] Stephen A Cook. An overview of computational complexity. *Commun. ACM*, 26(6):400–408, 1983.

[78] Richard M Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. ACM*, 32(4):762–773, 1985.

[79] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

[80] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J.Comput.*, 15(4):1036, 1986.

[81] Mark Goldberg. Parallel algorithms for three graph problems. *Congressus Numerantium*, 54:111–121, 1986.

[82] Andrew Goldberg, Serge Plotkin, and Gregory Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 315–324, 1987.

[83] Mark Goldberg and Thomas Spencer. A new parallel algorithm for the maximal independent set problem. *SIAM J.Comput.*, 18(2):419–427, 1989.

[84] Mark Goldberg and Thomas Spencer. Constructing a maximal independent set in parallel. *SIAM J. Discrete Math.*, 2(3):322–328, 1989.

[85] Don Coppersmith, Prabhakar Raghavan, and Martin Tompa. Parallei graph algorithms that are efficient on average. *Inf. Comput.*, 81(3):318–333, 1989.

[86] Kayhan. Erciyes. *Distributed graph algorithms for computer networks*. Springer Science Business Media, London, 2013.

[87] Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi, and Akka Zemmari. An optimal bit complexity randomized distributed mis algorithm. *Distrib. Comput.*, 23:331–340, 2011.

[88] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 270–277. SIAM, 2016.

[89] Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–820. SIAM, 2019.

[90] Baruch Awerbuch, Andrew V Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science*, volume 42, pages 364–369, Research Triangle Park, NC, USA, 1989. IEEE.

[91] Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 581–592. Association for Computing Machinery New York, NY, United States, 1992.

[92] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distrib Comput*, 14(2):97–100, 2001.

[93] Fabian Kuhn, Thomas Moscibroda, Tim Nieberg, and Roger Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In *Distributed Computing: 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005. Proceedings 19*, pages 273–287. Springer, 2005.

[94] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. 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*, pages 308–317, 2012.

[95] Ozkan Arapoglu, Vahid Khalilpour Akram, and Orhan Dagdeviren. An energy-efficient, self-stabilizing and distributed algorithm for maximal independent set construction in wireless sensor networks. *Computer Standards & Interfaces*, 62:32–42, 2019.

[96] Nabil Kadjouh, Ahcène Bounceur, Abdelkamel Tari, Loïc Lagadec, Reinhardt Euler, and Madani Bezoui. A new leader election algorithm based on the wbs algorithm dedicated to smart-cities. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, pages 1–5, 2019.

[97] Nabil Kadjouh, Ahcène Bounceur, Madani Bezoui, Mohamed Essaid Khanouche, Reinhardt Euler, Mohammad Hammoudeh, Loïc Lagadec, Sohail Jabbar, and Fadi Al-Turjman. A dominating tree based leader election algorithm for smart cities iot infrastructure. *Mobile Netw Appl*, 28:718–731, 2023.

[98] S.P. Borgatti. Identifying sets of key players in a social network. *Comput Math Organiz Theor*, 12:2134, 2006.

[99] A Arulselvan, C.W Commander, L Elefteriadou, and P. M. Pardalos. Detecting critical nodes in sparse graphs. *Computers and Operations Research.*, 36(7):2193–2200, 2009.

[100] C.J Kuhlman, V.A Kumar, M.V Marathe, S Ravi, and D.J. Rosenkrantz. Finding critical nodes for inhibiting diffusion of complex contagions in social networks. *Machine Learning and Knowledge Discovery in Databases.*, 2010.

[101] D.M Scott, D.C Novak, L Aultman-Hall, and F. Guo. Network robustness index: A new method for identifying critical links and evaluating the performance of transportation networks. *Journal of Transport Geography.*, 14(3):215–227, 2006.

[102] M Guettiche and H. Kheddouci. Critical links detection in stochastic networks: application to the transport networks. *International Journal of Intelligent Computing and Cybernetics.*, 12(1):42–69, 2019.

[103] ASHWIN Arulselvan, Clayton W Commander, Panos M Pardalos, and OLEG Shylo. Managing network risk via critical node identification. *Risk management in telecommunication networks, Springer*, pages 79–92, 2007.

[104] ASHWIN Arulselvan, Clayton W Commander, Panos M Pardalos, and OLEG Shylo. Managing network risk via critical node identification. *Risk management in telecommunication networks, Springer*, pages 79–92, 2007.

[105] Sergey M Senderov and Sergey V Vorobev. Approaches to the identification of critical facilities and critical combinations of facilities in the gas industry in terms of its operability. *Reliability Engineering & System Safety*, 203:107046, 2020.

[106] B Addis, M Di Summa, and A. Grosso. Identifying critical nodes in undirected graphs: complexity results and polynomial algorithms for the case of bounded treewidth. *Discrete Appl. Math.*, 161(16):2349–2360, 2013.

[107] M Di Summa, A Grosso, and M. Locatelli. Complexity of the critical node problem over trees. *Comput.Oper. Res.*, 38(12):1766–1774, 2011.

[108] A Berger, A Grigoriev, and R. Zwaan. Complexity and approximability of the k-way vertex cut. *Networks.*, 63(2):170–178, 2014.

[109] S Shen, J.C Smith, and R. Goli. Exact interdiction models and algorithms for disconnecting networks via node deletions. *Discrete Optim.*, 9(3):172–188, 2012.

[110] S Shen and J.C. Smith. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. *Networks.*, 60(2):103–119, 2012.

[111] D.T Guyen, Y Shen, and M.T. Thai. Detecting critical nodes in interdependent power networks for vulnerability assessment. *IEEE Trans. Smart Grid*, 4(1):151–159, 2013.

[112] R Aringhieri, A Grosso, P Hosteins, and R. Scatamacchia. A general evolutionary framework for different classes of critical node problems. *Eng. Appl. Artif. Intell.*, 55:128–145, 2016.

[113] T.N Dinh and M.T. Thai. Network under joint node and link attacks: vulnerability assessment methods and analysis. *IEEE/ACM Trans. Netw.*, 23(3):1001–1011, 2015.

[114] A Arulselvan, C.W Commander, O Shylo, and P.M. Pardalos. Cardinality-constrained critical node detection problem. In *In: Gülpınar, N., Harrison, P., Rüstem, B. (eds.) Performance Models and Risk Management in Communications Systems,*, page 79–91, New York, 2011. Springer.

[115] M Lalou, M.A Tahraoui, and H. Kheddouci. component-cardinality-constrained critical node problem in graphs. *Discrete Appl. Math.*, 210(3):150–163, 2016.

[116] B Addis, M Di Summa, and A.s Grosso. Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth. *Discrete Applied Matehmatics.*, 161:2349–2360, 2013.

[117] Y Shen, N. P Nguyen, Y. Xuan, and M. T. Thai. On the discovery of critical links and nodes for assessing network vulnerability. *in IEEE/ACM Transactions on Networking.*, 21(3):963–973, 2013.

[118] P Hosteins, R Rosario Scatamacchia, A Grosso, and R. Aringhieri. The connected critical node problem. *Theor. Comput. Sci.*, 923:235–255, 2022.

[119] M Di Summa and S.M.O. Faruk. Critical node/edge detection problems on trees. *Q J Oper Res.*, 40, 2022.

[120] Abdelmoujib Megzari, PV Pravija Raj, Walid Osamy, and Ahmed M Khedr. Applications, challenges, and solutions to single-and multi-objective critical node detection problems: a survey. *The Journal of Supercomputing*, 79(17):19770–19808, 2023.

[121] Niu Z, Li Q, Ma C, Li H, Shan H, and Yang F. Identification of critical nodes for enhanced network defense in manet-iot networks. *IEEE Access.*, 8:183571–183582, 2020.

[122] Shailendra Shukla. Angle based critical nodes detection (abcnd) for reliable industrial wireless sensor networks. *Wireless Personal Communications*, 130(2):757–775, 2023.

[123] Ishfaq Ahmad, Addison Clark, Muhammad Ali, Hansheng Lei, David Ferris, and Alex Aved. Determining critical nodes in optimal cost attacks on networked infrastructures. *Discover Internet of Things*, 4(1):2, 2024.

[124] Mohammed Lalou and Hamamache Kheddouci. A polynomial-time algorithm for finding critical nodes in bipartite permutation graphs. *Optimization Letters*, 13:1345–1364, 2019.

[125] S. Fortunato. Community detection in graphs. *Phys. Rep.*, 486(3):75–174, 2010.

[126] Vera Tomaino, Ashwin Arulselvan, Pierangelo Veltri, and Panos M Pardalos. Studying connectivity properties in human protein–protein interaction network in cancer pathway. *Data Mining for Biomarker Discovery*, pages 187–197, 2012.

[127] Mohammed Lalou and Hamamache Kheddouci. Least squares method for diffusion source localization in complex networks. In *International Workshop on Complex Networks and their Applications*, pages 473–485. Springer, 2016.

[128] Mohammed Lalou, Hamamache Kheddouci, and Salim Hariri. Identifying the cyber attack origin with partial observation: a linear regression based approach. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 329–333. IEEE, 2017.

[129] Mohammed Lalou and Hamamache Kheddouci. Network vulnerability assessment using critical nodes identification. In *2023 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2023.

[130] Ashwin Arulselvan, Clayton W Commander, Oleg Shylo, and Panos M Pardalos. Cardinality-constrained critical node detection problem. *Performance models and risk management in communications systems*, pages 79–91, 2011.

[131] W Pullan. Heuristic identification of critical nodes in sparse real-world graphs. *J.Heuristics*, 21(5):577–598, 2015.

[132] E Balas and C.C. de Souza. The vertex separator problem: a polyhedral investigation. *Math. Program.*, 103(3):583–608, 2005.

[133] Chanjuan Liu, Shike Ge, and Yuanke Zhang. Identifying the cardinality-constrained critical nodes with a hybrid evolutionary algorithm. *Information Sciences*, 642:119140, 2023.

[134] M Ventresca and D. Aleman. A randomized algorithm with local search for containment of pandemic disease spread. *Comput. Oper. Res.*, 48(3):11–19, 2014.

[135] Mohammed Lalou and Hamamache Kheddoucl. Finding important nodes in chordal graphs. In *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1448–1452. IEEE, 2024.

[136] Gabor Kecskemeti, Giuliano Casale, Devki Nandan Jha, Justin Lyon, and Rajiv Ranjan. Modelling and simulation challenges in internet of things. *IEEE cloud computing*, 4(1):62–69, 2017.

[137] Reham Almutairi, Giacomo Bergami, and Graham Morgan. Advancements and challenges in iot simulators: A comprehensive review. *Sensors*, 24(5):1511, 2024.

[138] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE communications surveys & tutorials*, 20(1):416–464, 2017.

[139] Ehizojie Ojie and Ella Pereira. Simulation tools in internet of things: a review. In *Proceedings of the 1st international conference on internet of things and machine learning*, pages 1–7, 2017.

[140] Charbel Mattar, Jacques Bou Abdo, Jacques Demerjian, and Abdallah Makhoul. Network diffusion algorithms and simulators in iot and space iot: A systematic review. *Journal of Sensor and Actuator Networks*, 14(2):27, 2025.

[141] Cupcarbon network simulator. Available at: https://cupcarbon.com/.

[142] Cupcarbon user guide. Available at: https://freenwork.com/cupcarbon/cupcarbon_user_guide.pdf.

[143] Kamal Mehdi, Massinissa Lounis, Ahcène Bounceur, and Tahar Kechadi. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 126–131, Lisbon, Portugal, 2014. Institute for Computer Science, Social Informatics and Telecommunications Engineering (ICST).

[144] Ramandeep Gill and Tarun Kumar Dubey. Shortest path computation technique in wsn for iot applications. *25 February 2022, PREPRINT (Version 1) available at Research Square*, 2022.

[145] Cristina Lopez-Pavon, Sandra Sendra, and Juan F Valenzuela-Valdés. Evaluation of cupcarbon network simulator for wireless sensor networks. *Network Protocols and Algorithms*, 10(2):1–27, 2018.

[146] Madani Bezoui, Ahcène Bounceur, Reinhardt Euler, and Moulaï Mustapha. A new distributed algorithm for finding dominating sets in iot networks under multiple criteria. In *12th International Conference on Multiple Objective Programming and Goal Programming (MOPGP 2017)*, 2017.

[147] Ali Lalouci and Zoubeyr Farah. A distributed algorithm to critical node identification in iot networks. *Journal of Communications Software and Systems*, 21(1):43–52, 2025.

[148] Ali Lalouci and Farah Zoubeyr. A distributed algorithm for secure dominating set problem in iot networks. *Studies in Engineering and Exact Sciences*, 5(2):01–25, 2024.

Abstract

The increasing complexity of modern networks, particularly in the context of the Internet of Things (IoT), has necessitated the development of advanced methods to protect these systems against various security problems. Graph theory has emerged as a powerful tool for modeling and solving network security problems due to its direct relevance to network structures. A network can be represented as a set of nodes connected by physical or logical links, and graph parameters such as the dominating set, secure dominating set, and critical node set are critical for optimizing network protection. The Dominating Set helps in selecting nodes to ensure information transmission within the network, while the secure dominating set identifies the set of guards that protect the entire network, playing a crucial role in network security. The Critical node set allows for identifying key nodes whose removal maximally disconnects the network, enabling targeted defense strategies. Despite their importance, obtaining optimal values for these graph parameters is a challenging task, as they often involve NP-complete problems that require approximate solutions. This thesis aims to address this challenge by proposing distributed algorithms that efficiently approximate these graph parameters, making them suitable for the resource-constrained, and decentralized nature of IoT networks. By leveraging distributed computing techniques, the research develops heuristic-based solutions for the secure dominating set, maximal independent set, and critical node set problems, focusing on IoT networks where traditional sequential methods are inadequate due to their computational complexity. The proposed algorithms are evaluated through simulations using the CupCarbon IoT simulator, demonstrating significant improvements in efficiency, scalability, and security. This work contributes both theoretical insights and practical solutions, enhancing the resilience of IoT networks while addressing the unique challenges posed by their distributed and dynamic nature.

Keywords: IoT networks protection, Graph parameters, Critical node sets, Secure dominating sets, NP-Complete problems, Distributed algorithms, Distributed systems.

Résumé

La complexité croissante des réseaux modernes, en particulier dans le contexte de l'Internet des Objets, a nécessité le développement de méthodes avancées pour protéger ces systèmes contre divers problèmes de sécurité. La théorie des graphes s'est imposée comme un outil puissant pour modéliser et résoudre les problèmes de sécurité des réseaux en raison de sa pertinence directe pour les structures des réseaux. Un réseau peut être représenté comme un ensemble de nœuds reliés par des liens physiques ou logiques, et des paramètres de graphe tels que l'ensemble dominant, l'ensemble dominant sécurisé et l'ensemble des nœuds critiques sont essentiels pour optimiser la protection du réseau. L'ensemble dominant aide à sélectionner les nœuds pour garantir la transmission des informations dans le réseau, tandis que le l'ensemble dominant sécurisé identifie l'ensemble des gardiens qui protègent l'ensemble du réseau, jouant ainsi un rôle crucial dans la sécurité du réseau. L'ensemble des nœuds critiques permet d'identifier les nœuds clés dont la suppression décompose au maximum le réseau, permettant ainsi des stratégies de défense ciblées. Malgré leur importance, obtenir des valeurs optimales pour ces paramètres de graphe est une tâche difficile, car elles impliquent souvent des problèmes NP-complets qui nécessitent des solutions approximatives. Cette thèse vise à relever ce défi en proposant des algorithmes distribués qui approchent efficacement ces paramètres de graphe, les rendant adaptés à la nature décentralisée et aux ressources limitées des réseaux IoT. En exploitant les techniques de calcul distribué, la recherche développe des solutions basées sur des heuristiques pour l'ensemble dominant sécurisé, l'ensemble indépendant maximal et le problème de détection des nœuds critiques, en se concentrant sur les réseaux IoT où les méthodes séquentielles traditionnelles sont inadéquates en raison de leur complexité computationnelle. Les algorithmes proposés sont évalués par des simulations utilisant le simulateur CupCarbon IoT, montrant des améliorations significatives en termes d'efficacité, de scalabilité et de sécurité. Ce travail apporte à la fois des éclairages théoriques et des solutions pratiques, renforçant la résilience des réseaux IoT tout en abordant les défis uniques posés par leur nature distribuée et dynamique.

Mots-clés : Protection des réseaux IoT, Paramètres de graphe, Ensembles de nœuds critiques, Ensembles dominants sécurisés, Problèmes NP-complets, Algorithmes distribués, Systèmes distribués.

الملخص

إن التعقيد المتزايد للشبكات الحديثة، وخاصة في سياق إنترنت الأشياء(IoT) ، قد استدعى تطوير أساليب متقدمة لحماية هذه الأنظمة ضد الهجمات المختلفة. لقد برزت نظرية الرسوم البيانية كأداة قوية لنمذجة وحل مشاكل أمان الشبكات بفضل علاقتها المباشرة بهياكل الشبكات. يمكن تمثيل الشبكة كمجموعة من العقد المتصلة بروابط فيزيائية أو منطقية، وتعد معلمات الرسم البياني مثل مجموعة الهيمنة، مجموعة الهيمنة الآمنة، ومجموعة العقد الحرجة أساسية لتحسين حماية الشبكة. تساعد مجموعة الهيمنة في اختيار العقد لضمان نقل المعلومات ضمن الشبكة، في حين أن مجموعة الهيمنة الآمنة تحدد مجموعة الحراس الذين يحمون الشبكة بالكامل، مما يلعب دورًا حيويًا في أمان الشبكة. تتيح مجموعة العقد الحرجة تحديد العقد الأساسية التي يؤدي حذفها إلى تفكيك الشبكة إلى أقصى حد، مما يتيح استراتيجيات دفاعية مستهدفة. على الرغم من أهميتها، فإن الحصول على قيم مثلى لهذه المعلمات البيانية يعد مهمة صعبة، حيث إنها غالبًا ما تتضمن مشكلات NP-complete التي تتطلب حلولًا تقريبية. تهدف هذه الأطروحة إلى معالجة هذا التحدي من خلال اقتراح خوارزميات موزعة تقرب هذه المعلمات البيانية بفعالية، مما يجعلها مناسبة للطبيعة اللامركزية والمحدودة الموارد لشبكات إنترنت الأشياء. من خلال الاستفادة من تقنيات الحوسبة الموزعة، تطور الدراسة حلولًا معتمدة على الخوارزميات الاستدلالية لمجموعة الهيمنة الآمنة، مجموعة العقد المستقلة القصوى، ومشكلة الكشف عن العقد الحرجة، مع التركيز على شبكات إنترنت الأشياء حيث تكون الأساليب التقليدية غير فعالة بسبب تعقيدها الحسابي. يتم تقييم الخوارزميات المقترحة من خلال محاكاة باستخدام محاكي CupCarbon لشبكات إنترنت الأشياء، مما يظهر تحسنًا كبيرًا في الكفاءة، والقابلية للتوسع، والأمان. يساهم هذا العمل في تقديم رؤى نظرية وحلول عملية، مما يعزز من مرونة شبكات إنترنت الأشياء بينما يتعامل مع التحديات الفريدة التي تفرضها طبيعتها الموزعة والديناميكية.

الكلمات المفتاحية: حماية شبكات إنترنت الأشياء، معلمات الرسوم البيانية، مجموعة العقد الحرجة، مجموعة الهيمنة الآمنة، مشاكلNP-complete ، الخوارزميات الموزعة، الأنظمة الموزعة.