People's Democratic Republic of Algeria
Ministery of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Borj Bou Arréridj
Faculty of Mathematics and Computer Science
Department of Computer Science

DISSERTATION
Presented in fulfillment of the requirements of obtaining the degree
Master in Computer Science
Specialty: Networking & Multimedia

THEME

Enhanced Data Collection by Strategically Selecting Visiting Points for Mobile Sink in Wireless Sensor Networks

Presented by:
Ichrak MADOUI
Ibtissem NOUIOUA

Publicly defended on: 11/06/2025
In front of the jury composed of:
President: Dr. Boubakeur MOUSSAOUI
Examiner: Dr. Oussama SENOUCI
Supervisor: Dr. Nadjib BENAOUDA

2024/2025

Dedication

With profound gratitude and boundless love, I dedicate this humble work to those who have been my unwavering pillars.

To my beloved family, my sanctuary and guiding stars, the wellspring of my greatest strength. Their enduring trust, infinite patience, and steadfast support have laid the very foundation upon which this journey was built. Without their love, this achievement would remain but a distant dream.

To my fiancé, whose constant support and reassuring presence have been my refuge and inspiration. His unwavering encouragement and belief in me have fuelled my perseverance through even the most challenging moments.

To my wonderful friends, who have stood steadfastly by my side throughout every step of this path. Their kindness, encouragement, and shared moments have enriched this journey, making it all the more meaningful.

Ibtissem NOUIOUA

Dedication

To those who stood by me, I dedicate this graduation.

*To the dearest one, to the one whose name I carry with pride and honor, who cleared the thorns from my path and planted comfort and tranquility in their place to my **father**. Thank you for being my greatest joy.*

*To the one who taught me values before I could even speak them, to the bridge that lifts me toward paradise, and to the one whose prayers never cease to carry my name day and night to my **mother**, my beloved.*

*To those whom God granted me the blessing of their presence, to the source of my strength, the sturdy wall around my heart my brothers, **Mohamed** and **Haythem**, and my sister, **Lina**.*

*To my **grandparents**, may God have mercy on them, whose words remain immortal and whose memories are never forgotten.*

*To my **family**, the source of my strength and endless encouragement.*

*To my best friend, my journey companion, and my sister **Aya**, who stood by me in every moment a support and strength beyond measure.*

*To the one with whom I shared unforgettable experiences and who has always encouraged me **Yasmine**, and to those who walked alongside me with their hearts and soulsmy friends: **Leila**, **Ibtissem**, **Imane**, **Bicha**, **Hanane**, **Aya**, **Wissem**, **Maria**, and **Manar**.*

*To steadfast **Gaza**, may Allah have mercy on its noble martyrs and grant them the highest place in Jannah. Be patient, O Gaza, for the promise of Allah is true.*

Ichrak MADOUI

Acknowledgment

With heartfelt thanks, we express our gratitude to Allah, the Almighty, for granting us the strength, patience, and perseverance needed to complete this work successfully.

To our esteemed supervisor, Mr. Nadjib BENAOUDA, we extend our heartfelt thanks. His wisdom, unwavering support, and invaluable guidance have been our beacon of light throughout this journey. His trust in our abilities and his constant encouragement inspired us to strive for excellence at every step.

Our profound appreciation goes to the professors of the Computer Science Department, whose dedication and expertise have shaped our academic growth. In particular, we would like to sincerely thank Dr. Boubakeur MOUSSAOUI, Dr. Oussama SENOUCI, Dr. Ali MOUSSAOUI, Dr. Farid NOUIOUA, Dr. Mourad NOUIOUA, and Dr. Sara BOUTOUHAMI, our professors in previous years, whose teachings, encouragement, and support have left a lasting impact on our academic journey. We are also thankful to our colleagues for their solidarity, encouragement, and the meaningful exchanges that accompanied us along this path.

Finally, we owe a boundless debt of gratitude to our families and friends. Their unwavering love, support, and encouragement have been the pillars that sustained us through every trial. It is through their presence that this achievement became possible.

Abstract

This study examines issues related to data collection in wireless sensor networks (WSNs) using mobile sinks. We focus on optimizing this collection process to improve energy-efficiency, extend network lifetime, and reduce data collection delay. To address these issues, we propose a novel approach called EDCVP, which stands for *Enhanced Data Collection via Visiting Points*, wherein the mobile sink adheres to a predetermined trajectory for data collection. EDCVP presents a structured method for selecting visiting points (VPs), locations where the mobile sink halts to collect data. These points are carefully selected to reduce the mobile sink's tour length and, hence, the data collection latency. We assessed the efficiency of the proposed protocol by implementing and comparing it in conjunction with the related protocol, EDEDA, using the NS-3 simulator. A series of experiments was done under different parameter settings. The simulation results demonstrate the effectiveness of our approach relative to EDEDA: EDCVP reduces energy consumption by 4%, prolongs network lifetime by 10.63%, minimizes the mobile sink's tour length by 88.72% and decreases data collection delay by 140.57%. These improvements confirm the efficiency of EDCVP in enhancing overall network performance in WSNs.

Keywords: Wireless sensor networks, mobile sink, visiting points, fixed-path collection, NS-3.

Résumé

Cette étude se penche sur les problématiques liées à la collecte de données dans les réseaux de capteurs sans fil en utilisant des puits mobiles. Notre attention se porte sur l'optimisation de ce processus de collecte afin d'améliorer l'efficacité énergétique, de prolonger la durée de vie du réseau et de réduire la période de collecte des données. Pour résoudre ces problèmes, nous proposons une nouvelle approche appelée EDCVP, qui signifie *Enhanced Data Collection Utilizing Visiting Points*, où le puits mobile suit une trajectoire prédéterminée pour la collecte de données. EDCVP propose une méthode structurée pour la sélection des points de visite, qui sont des emplacements où le puits fait une halte pour collecter des données. Ces points sont soigneusement sélectionnés afin de réduire la longueur du parcours du puits et, par conséquent, la latence de la collecte de données. L'efficacité du protocole proposé a été évaluée en l'implémentant et en le comparant avec le protocole connexe, EDEDA, en utilisant le simulateur NS-3. Plusieurs expériences ont été réalisées en modifiant les paramètres. Les résultats de la simulation démontrent l'efficacité de notre approche par rapport à EDEDA : EDCVP réduit la consommation d'énergie de 4%, prolonge la durée de vie du réseau de 10,63%, minimise la longueur du parcours du puits de 88,72% et diminue la durée de collecte de données de 140,57%. Ces améliorations confirment l'efficacité de EDCVP dans l'amélioration des performances globales du réseau.

Mots-clés: Réseaux de capteurs sans fil, puits mobile, points de visite, collecte sur chemin fixe, NS-3.

الملخص

تتناول هذه الدراسة مشكلات جمع البيانات في شبكات المستشعرات اللاسلكية (WSNs) باستخدام نقاط التجميع ، مع التركيز على تحسين كفاءة استهلاك الطاقة، إطالة عمر الشبكة، وتقليل زمن جمع البيانات. لمعالجة هذه التحديات، نقترح منهجية جديدة تُعرف باسم EDCVP (Enhanced Data Collection via Visiting Points) حيث يتبع الجامع المتحرك مسارا ثابتا لجمع البيانات من نقاط تجميع مختارة بعناية (VPs)، لتقصير طول جولة زمن التجميع. تم تقييم كفاءة البروتوكول المقترح عبر تنفيذه ومقارنته مع بروتوكول EDEDA باستخدام المحاكي NS-3، من خلال مجموعة تجارب باستخدام إعدادات مختلفة. أظهرت النتائج أن EDCVP يقلل من استهلاك الطاقة بنسبة 10.63%، ويُطيل عمر الشبكة بنسبة 4% ، ويقلل من طول جولة الجامع بنسبة 88.72%، ومن زمن جمع البيانات بنسبة 140.57%، مما يؤكد فعاليته في تحسين أداء الشبكة.

الكلمات المفتاحية: شبكات المستشعرات اللاسلكية، الجامع المتحرك، نقاط التجميع، جمع البيانات بمسار ثابت، NS-3.

Table of Contents

List of Abbreviations

WSN	Wireless Sensor Network
SN	Sensor Node
BS	Base Station
CH	Cluster Head
GCH	Grid Cell Head
RC	Rendezvous Cell
RGCH	Rendezvous Grid Cell Head
VP	Visiting Point
EDEDA	Energy-and Delay-Efficient Data Acquisition
IoT	Internet of Things
MAC	Medium Access Control
MWSN	Mobile Wireless Sensor Network
MS	Mobile Sink
NS-3	Network Simulator 3
QoS	Quality of Service
TDMA	Time Division Multiple Access
UMV	Unmanned Maritime Vehicle
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
RE	Residual Energy

List of Figures

List of Tables

List of Algorithms

Chapter 1

General Introduction

1.1 Background and context

Wireless Sensor Networks (WSNs) consist of spatially distributed autonomous sensor nodes (SNs) used to monitor physical or environmental conditions like temperature, sound, or pressure. These SNs collaboratively gather and sent the observed data to a specified entity generally referred to as a base station or sink, where the data might undergo additional analysis. WSNs have been incorporated into the modern network infrastructures, and constitute a key aspect of the adoption of the Internet of Things (IoT) as they are primarily responsible for data collecting from the physical environment. WSNs are widely adopted across various domains, including the IoT, smart agriculture, industrial automation, healthcare monitoring, environmental surveillance, and smart cities [1]. Given all their versatility, they can become a complete weapon for rapid data collection and wise decision-making.

However, the biggest challenge in WSNs has always been energy efficiency, given that sensor nodes are primarily battery-powered, particularly in traditional architectures including static sinks. In such designs, energy efficiency is a significant obstacle for WSNs. This configuration causes the well-known hotspot issue [2] and shortens network lifetime when sensor nodes closest to the sink deplete their energy quicker from too many relay transmissions.

Mobile sinks (MSs) have emerged as a promising alternative to resolve this issue. MSs can take several forms depending on the specific applications they are used for. Typical setups include Unmanned Aerial Vehicles (UAVs), Unmanned Ground Vehicles (UGVs) or Unmanned

Maritime Vehicles (UMVs). Furthermore, they can also refer to devices that are transported by humans or animals, depending on the particular use case. An MS can significantly reduce energy consumption at critical nodes, balance energy usage across the network, and extend the overall network lifetime by traversing the network and collecting data from various locations.

An MS can significantly reduce energy consumption at critical nodes, balance energy usage across the network, and extend the overall network lifetime by traversing the network and collecting data from various locations. For example, a large-scale WSN can be deployed in an agricultural field to monitor soil moisture, temperature, and crop health. The network consists of many static SNs distributed around the area. A mobile drone serves as the principal sink for data collection, replacing a fixed central sink. This drone possesses communication capabilities to regularly traverse the agricultural field, establish links with the SNs, and gather their aggregated data. Similarly, in a forest fire scenario, a WSN can be deployed to detect potential fire occurrences using a drone. Indeed, The drone's mobility diminishes energy consumption by eliminating need for continuous long-range communication from SNs and allows efficient data collection from various field sectors without depending exclusively on multi-hop transmissions.

1.2 Problem statement

In this final year research project, we are interested in the challenges related to data collection using MSs. This study focuses on application scenarios that require the regular collection of data by an MS from all nodes inside the network. Two possible approaches may be considered for accomplishing this collection. The initial solution entails the MS visiting each SN inside the network. In this case, the duration of collection will be significant due to the considerable distance the sink has to travel to reach all nodes in the network. The second alternative involves designating certain locations within the deployment area or nodes within the network as visiting points (VPs). The MS must thereafter visit these VPs to collect data from nodes adjacent to these locations. The efficiency of this second method, however, depends on two critical factors:

1. **The number of VPs selected:** While too few VPs might result in energy inefficiencies, more VPs lengthen collecting durations.

2. **The path taken by the MS:** In order to minimize collecting delay, the total distance

traveled should be kept to a minimum.

However, determining the optimal set of VPs and finding the best trajectory for the MS are complex tasks due to several challenges:

- The number of VPs must be minimized while ensuring that all nodes can offload their data efficiently.
- VPs must be chosen judiciously to reduce transmission distances and SN energy usage.
- The MS trajectoty must be optimized in order to ensure the shortest possible path while visiting each VP precisely once.

Addressing these challenges requires an efficient approach that balances energy consumption, network lifetime and collection latency. This study aims to develop a data collection strategy that optimizes both VPs selection and MS movement, ensuring a scalable and energy-efficient solution for faster data collection in WSNs.

1.3 Objectives and contributions

This study offers a novel data-collecting method for WSNs named EDCVP (Enhanced Data Collection via Visiting Points) to handle the previously mentioned issues. The major objectives are to reduce data collection delay, improve energy efficiency in data acquisition, and extend network lifetime. EDCVP judiciously selects specific VPs where the MS can halt and gather data from adjacent SNs, rather than necessitating visits to all of them. Our proposed solution minimizes the number of stops the MS must make to cover all nodes in the network, hence decreasing data collecting latency. Moreover, EDCVP was designed to optimize energy consumption during data transmission and extend network longevity. The key features and contributions of the proposed EDCVP protocol are summarized as follows:

- EDCVP uses a structured virtual grid to divide the sensor area into grid cells. The size of each cell depends on the communication range of SNs and the deployment area's size. This configuration allows for organized clustering and localized data handling.
- Within each grid cell, a Grid Cell Head (GCH) is selected on the basis of its residual energy and proximity to the cell center. These GCHs are responsible for collecting and

aggregating data from their cell nodes. A subset of them is then promoted to Rendezvous Grid Cell Heads (RGCHs), which collect additional data from surrounding GCHs.

- A carefully limited number of VPs are placed at the intersections of four neighboring cells, enabling the MS to collect data from key positions with minimal stops and communication overhead. Each VP is placed in a way that allows the MS to communicate with multiple RGCHs using a single hop. The RGCHs in turn collect data from surrounding GCHs.

- The mobile sink's path determination problem is formulated as a Hamiltonian cycle, with a backtracking technique utilized as the solution method to calculate a valid path that visits all VPs exactly once.

1.4 Overview of validation methodology and key results

The performance of the proposed EDCVP protocol was evaluated using a series of simulations performed using the NS-3 network simulator [3]. The EDEDA [4] benchmark protocol was implemented under the same conditions for comparative analysis. The assessment was carried out on many network configurations to ensure equity and consistency, generating representative results, including performance graphs and visual snapshots. The evaluation focused on many key performance metrics, including overall energy consumption, network lifetime, data collection delay, MS's tour length, and the required number of VPs. The simulation results indicate that EDCVP meets its design goals and has higher performance compared to EDEDA.

1.5 Structure of the report

To provide a clear and coherent understanding of our research approach, this report is structured to gradually introduce the essential concepts related to our study, detail the proposed contribution, and present the evaluation results obtained through simulation. It is organized as follows:

- **Chapter 2** provides foundational knowledge about WSNs, covering their architecture, operational principles, communication mechanisms, major challenges, and a classifica-

tion of routing strategies.

- **Chapter 3** examines the use of MSs for data collection in WSNs. It explores the various forms of mobility relevant to such networks, highlights the benefits and challenges associated with the use of MSs, and reviews key contributions from the literature that have addressed similar problems to ours.

- **Chapter 4** outlines the proposed data collection protocol, EDCVP. It presents the system model and describes how the sensing area is organized using a structured grid, how GCHs are elected, and how RGCHs are choosen. It also details the mechanism adopted for selecting VPs and planning the mobile sink's path.

- **Chapter 5** is dedicated to the evaluation of EDCVP. It describes the simulation setup using the NS-3 simulator, explains the comparison methodology, and discusses the results obtained with respect to performance metrics such as energy efficiency and collection delay.

- This report ends with a **general conclusion** that summarize our contributions, identifies certain limitations of our work, and proposes future directions for enhancement.

- For completeness, the main C++ header files of the implemented protocol EDCVP together with the simulation setup file, are included in the appendix.

Chapter 2

Fundamentals of Wireless Sensor Networks

2.1 Introduction

WSNs have become a pivotal technology for enabling real-time monitoring and data collection in various physical environments. This chapter provides an overview of WSNs by exploring their core architecture, deployment scenarios for SNs, and communication schemes within the network. It also discusses key challenges faced by WSNs like limited energy resources, fault tolerance, and network reliability along with the primary energy conservation techniques like duty cycling, data aggregation, and cross-layer design. In addition, it introduces various routing protocols tailored to WSNs, enhancing energy efficiency and ensuring reliable data transmission. The chapter concludes by highlighting the wide-ranging applications of WSNs, particularly in remote or difficult-to-access environments, for improved decision-making and monitoring.

2.2 Definition

A WSN is a network composed of spatially distributed, autonomous SNs equipped with sensing, computation, and wireless communication capabilities. These nodes are configured to work cooperatively to monitor, collect, record, and transmit environmental data (such as temperature, humidity, or pressure) to a central destination, known as a *sink* or *base station*

(BS) using wireless technology [5, 6]. The WSN architecture is illustrated in Figure 2.1.

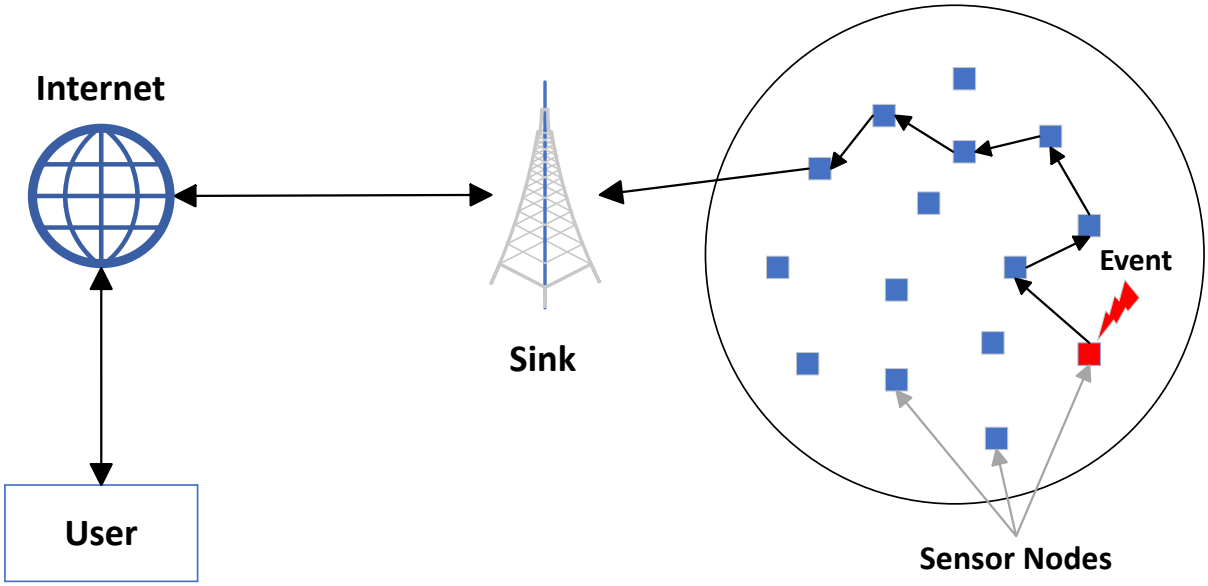


Figure 2.1: Architecture of a WSN [5].

2.3 Architecture of a SN

A SN measures the variation in current conditions in its surrounding environment using its sensor(s). The Analog-to-Digital Converter (ADC) unit transforms these measurements into relative electric signals, which the node's processor then processes. The node can wirelessly send the data generated by its processor to other nodes or to a designated sink point, via its transceiver. A sensor node may additionally be equipped with set of actuators and a location-finding system, depending on the specifications of the intended application [1].

In summary, to perform its functions, a SN comprises the following components:

1. A sensing system for data acquisition.
2. Data processing system.
3. A communication system for disseminating information.
4. Optionally, a set of actuators to perform actions.
5. Optionally, a location finding system to determine the node's position.

The architecture of a SN is shown in Figure 2.2.

2.4 SN functions

SNs in WSNs can operate as both data originators and data routers. In particular, they fulfill mainly two principal functions [6]:

1. **Sensing function:** They gather data from their environment by detecting physical phenomena such as temperature and humidity and monitoring specific areas like forests and wildlife habitats.

2. **Communication function:** They forward their own data and participate in forwarding the packets received from other nodes to the next destination in the multi-hop path to the sink.

2.5 Sink role

A sink node in a WSN is a BS or central node to which all data gathered by SNs is intended to be transferred. It acts as the main hub for obtaining, combining, and occasionally sending the environmental data gathered by SNs to an outside system for additional analysis [5, 6].

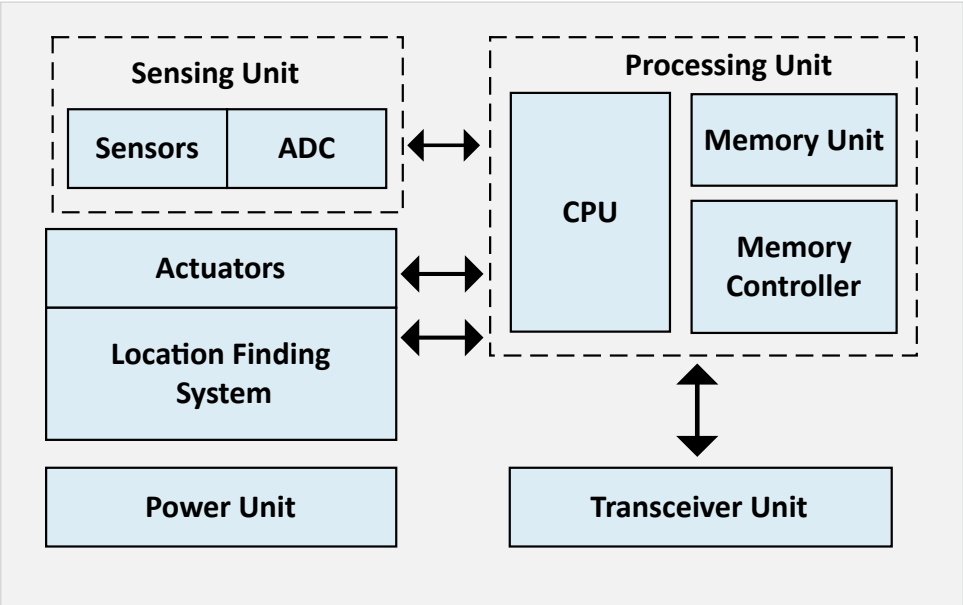


Figure 2.2: Architecture of a SN [1].

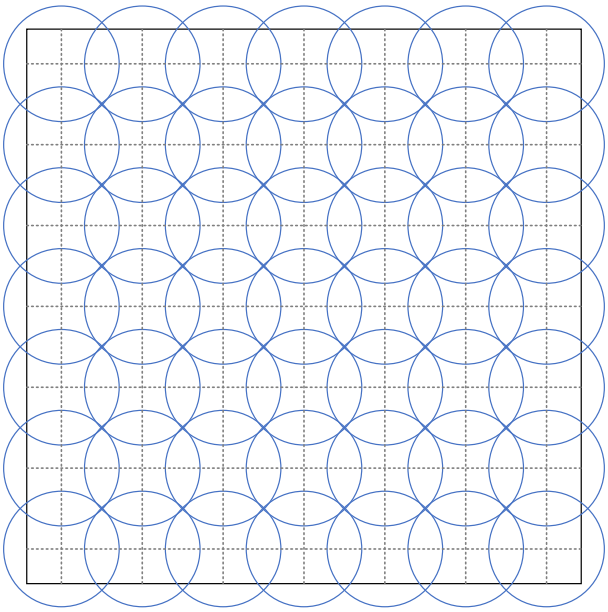


Figure 2.3: Deterministic deployment of wireless SNs in WSN [7].

2.6 Deployment scenarios for SNs

The deployment problem involves the placement of SNs in the sensor field to monitor targets and send data to destination nodes. There are two main methods [7, 8]: deterministic deployment and random deployment.

2.6.1 Deterministic deployment

Deterministic deployment involves predetermining node coordinates before dispersion, aiming to maximize target monitoring, minimize network costs, ensure responsible energy use, and extend sensor network lifetime. Figure 2.3 denotes the deterministic coverage of the nodes in WSN.

2.6.2 Random deployment

Certain applications may not support deterministic approaches due to hostile environments or high deployment costs. In this case, a random deployment can be used, with which the nodes are positioned arbitrarily without prior determination of their locations. In general, a WSN with randomly deployed SNs will perform worse than a deterministically distributed SNs. Figure 2.4 denotes the deterministic coverage of the nodes in a WSN [9].

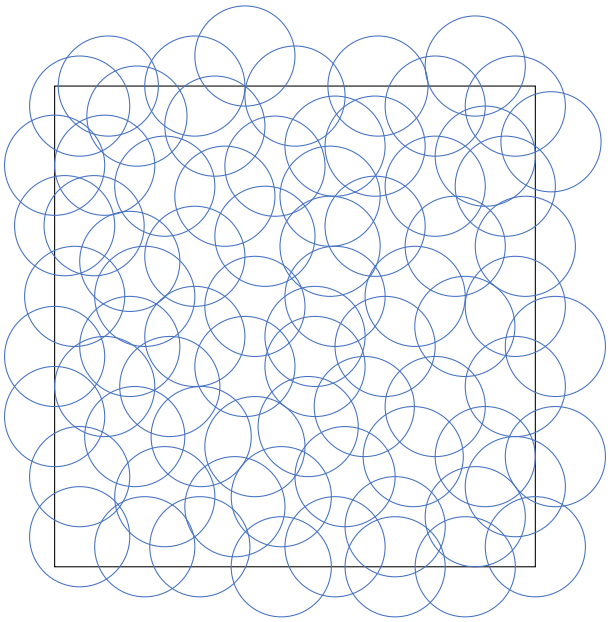


Figure 2.4: Random deployment of wireless SNs in a WSN [7].

2.7 Communication schemes in WSNs

In WSNs, sensors can be grouped according to their deployment context and behavior. These sensors can communicate with each other or with the sink in two different ways: periodic and nonperiodic.

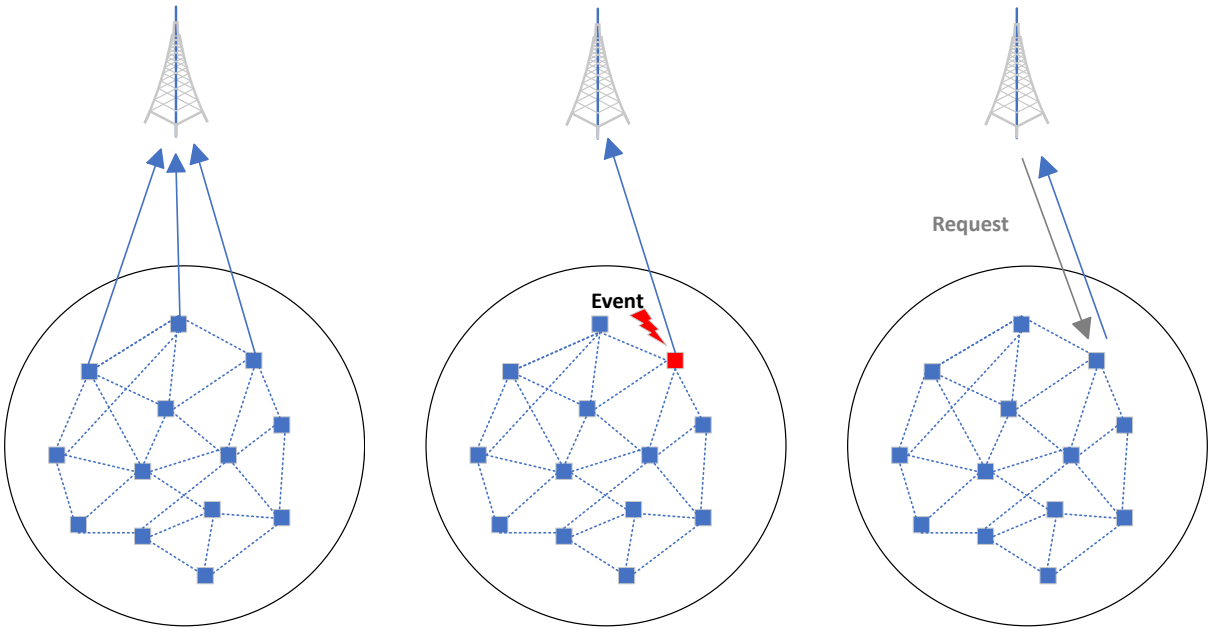
2.7.1 Periodic-based

In this case, the sensors consistently transmit data at regular intervals, even in the absence of specific events. Periodic sensing applications are mainly used for monitoring purposes, and they provide information about the environment in which the SNs are deployed. It is used in many fields, such as environmental monitoring, home automation, military surveillance, healthcare monitoring systems, etc [10].

2.7.2 Nonperiodic-based

Anytime an event occurs or the sink generates a query, the nodes respond by sending the sensed data [10]. These systems can be categorized mainly into two primary forms: event-based and query-based.

- **Event-based:** In these applications, SNs transmit detected data in response to event occurrences. Some nodes in the WSNs are typically placed in sleep mode to conserve en-



(a) Periodic-based (b) Event-based (c) Query-Based

Figure 2.5: Different communication schemes in WSNs [12].

ergy, while others remain awake to monitor for potential events. The nodes in sleep mode can be activated for communication. A trigger is required for that purpose. Some systems employ the wake-up radio idea to activate these SNs. Examples of event detection-based systems encompass several control systems and disaster management frameworks.

- **Query-based:** There are certain applications where users are occasionally presented with a large amount of data that may be redundant. For example, environmentalists would sometimes look for sensor data that meet their needs. They transmit the request to the sink, which then causes the SNs in that area to become active and start supplying the requested data [11].

The Figure 2.5 below illustrates the various communication schemes used in WSNs.

2.8 WSNs challenges

The unique features of WSNs provide several design challenges, with the main ones summarized as follows [13]:

1. The limited processing capabilities and battery limitations of SNs pose new challenges in hardware and software development, necessitating further research in hardware, network,

and communication protocol design.

2. WSNs consist of numerous nodes randomly or manually deployed over hostile areas, requiring automatic arrangement for communication and network establishment before information exchange.

3. SNs in unmanaged and dynamic environments are prone to failure due to battery depletion or harsh environmental conditions, rendering fault tolerance a key consideration in WSN application protocols.

4. Numerous WSN applications are constantly evolving, each with unique application requirements, necessitating the development of diverse network designs, communication protocols, and methods.

2.9 Objectives and design issues of WSNs

Several design goals must be met during the development of any solution intended for WSNs. These goals vary depending on the intended application. The primary objectives are as follows [13, 4]:

- WSN's main constraint is network lifetime, requiring low energy consumption and a balance in that consumption. In scenarios where SNs cannot be replaced, such as in environmental applications, this issue is of paramount importance.

- In some applications, such as environmental ones, SNs may cease to function due to malfunctions and physical damage caused by the harsh conditions in which the sensors are deployed. Fault tolerance in WSN environmental applications where the network is organized in a hierarchical manner can be achieved through re-clustering or rotating the role of cluster-head (CH) among nodes in the network. Moreover, these last two techniques aid in load balancing. Indeed, even load distribution among the nodes is required for significant communication in order to extend the network lifetime and avoid the hot spot problem.

- Redundancy can be decreased in systems where SNs are randomly placed by aggregating redundant data using functions like max, min or average. As a result, there is less local

duplication, less energy use, and less network traffic.

- SNs use direct (Single hop) or indirect (Multi-hop) transmission to send data to the sink. Communication can occur through many forms, such as broadcast, point-to-point or local way transmission. Depending on the application and as outlined in section 2.7, data transmission may occur at regular intervals (periodically), be initiated by certain events (event-driven), or occur in response to direct queries from the sink (query-based).

- Real-time communication is crucial in time-sensitive WSNs applications, such as industrial control or disaster response. The network must ensure constrained latency for data transmission to facilitate prompt and precise decision-making.

2.10 Types of energy consumption in SN

SNs have limited power sources and in many cases, cannot replenish power, making WSN lifetime heavily dependent on battery life. Each node plays two roles: data originator, gathering environmental data, and data router, relaying neighboring information. Due to low-power communication techniques, node communication range is limited, and multi-hop communication is required for large networks. SNs receive and forward neighboring data according to routing decisions, ensuring efficient energy consumption and network lifetime. SNs detect events, process data, and transmit it, dividing power consumption into sensing, data processing, and communication domains, performed by sensors, CPU, and radio, respectively [12].

Figure 2.6 illustrates the energy consumption levels of various components within a SN. Specifically, node energy can be consumed in:

1. **Sensing:** The type of application and the particular sensors being utilized determine the sensing power. It's possible that sporadic sensing uses less energy than constant event monitoring. The degree of event detection complexity is another important factor in figuring out energy usage.

2. **Communication:** A SN uses the most energy for data communication out of all three domains. The transceiver circuitry communicates when data is being received and transmitted. Furthermore, whenever the SN is not in need of transmitting or receiving data, a sizable amount of energy can be conserved by switching the transceiver to a sleep state.

3. **Data processing:** Data processing uses about the same amount of energy as sensing. On the other hand, computation uses a lot less energy than data transfer. The significant distinction between computing and communication highlights the role that local data processing plays in reducing power usage in multi-hop sensor networks.

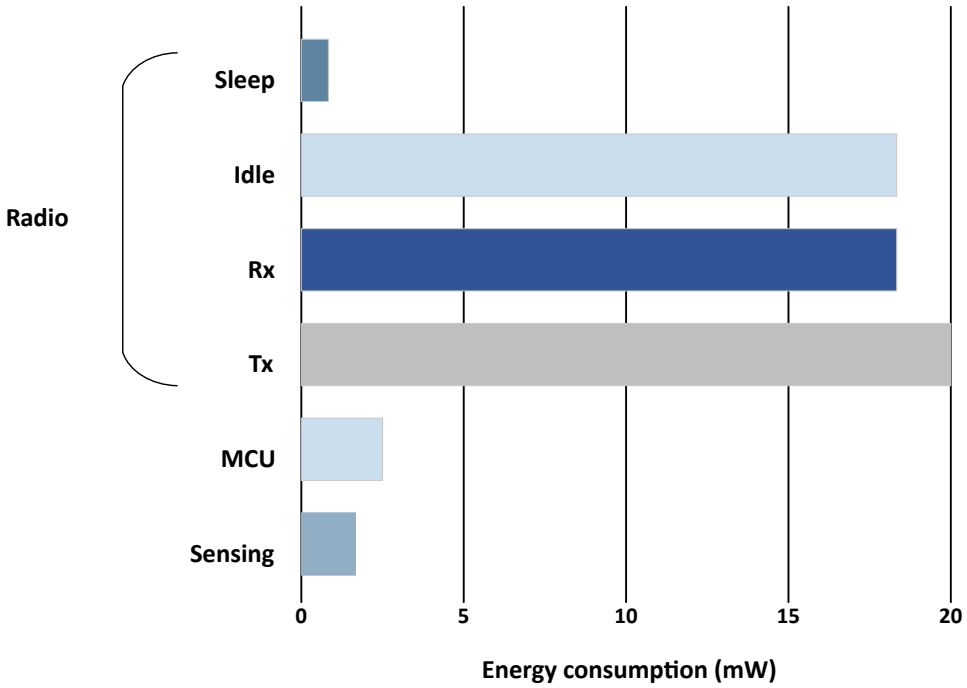


Figure 2.6: Energy consumption by different units of a SN [12].

2.11 Energy conservation methods

Reducing the energy consumption of SNs can extend the lifetime of a WSN. The following lists the several methods for lowering the energy usage of the network's SNs [14, 15].

- **Duty cycle approach:** Duty cycling is a method of conserving energy in WSNs by switching between active and sleep modes based on network activity. SNs coordinate their sleep or wake-up times, and any duty cycling scheme should be accompanied by a distributed sleep/wake-up time scheduling algorithm.

- **Routing:** In multi-hop communication, routing is an expensive and important phenomenon that uses a lot of energy. Efficient routing techniques are crucial for minimizing energy usage and prolonging network lifetime.

- **Medium access control (MAC) protocols:** The MAC protocols need to be energy-

efficient in order to lower the energy consumption and prolong the lifetime of WSN in addition to the routing protocols. The four primary causes of MAC energy waste are control packet overhead, overhearing, collision, and idle listening.

- **Data aggregation:** Data aggregation is another method that can lower energy usage in WSNs. Because WSN nodes are deployed in a dense manner, the data they detect is highly correlated. As a result, the detected data has a significant degree of redundancy. The goal of data aggregation is to lessen unnecessary data before sending.

- **Cross-layer design:** Cross-layer network designs can greatly enhance network performance, particularly energy efficiency, in situations when traditional layered protocol designs fall short of expectations.

- **Error control code (ECC):** A crucial component of any communication system is reliability. Random noise and channel fading can affect any radio broadcast. Error control mechanisms must thus be used in some way to mitigate these effects and raise system dependability [16]. Furthermore, ECC improves energy conservation by permitting acceptable bit error rates (BER) at diminished signal-to-noise ratios (SNR), enabling sensor nodes to operate at lower power levels during transmission. This benefit is used to reduce energy usage in WSNs [17].

- **Mobility-based:**

  Mobility-oriented methods, like the deployment of mobile sinks, mitigate the necessity for extensive data transmission distances. For instance, by dynamically repositioning the sink to proximity with data-generating regions. This reduces the energy used for distant nodes to transport their data, hence lowering the overall transmission cost, particularly in multi-hop situations [18, 1].

2.12 Routing protocols in WSNs

Routing in WSNs is crucial for energy efficiency, scalability, and reliability. Traditional protocols are not suitable due to resource limitations and sensor network requirements. Specialized protocols have been developed to minimize energy consumption, extend network lifetime, and adapt to dynamic conditions [19, 14]. Routing protocols in WSNs can be classified

in different ways. In this section, we present the classification provided in reference [14] which is illustrated in Figure 2.7.

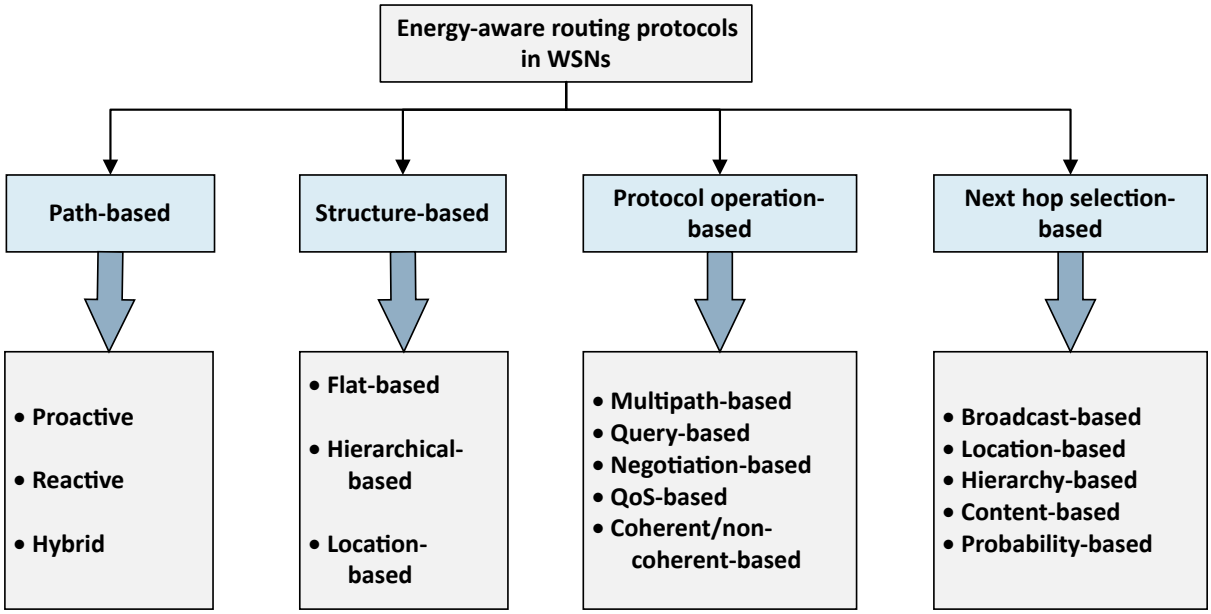


Figure 2.7: Classification of energy-aware routing protocols in WSNs [14].

The following subsections present the various considered categories of routing protocols in WSNs, including path-based, structure-based, protocol operation-based, and next hop selection-based protocols.

2.12.1 Path-based routing protocols

Path based routing protocols can be proactive, reactive or hybrid [20, 21]:

- Proactive routing protocols keep an up-to-date list of destinations and their corresponding routes in a table format. This is accomplished by periodically distributing the updated routing tables across the network.
- In reactive protocols, routes are created on demand. Route request packet flooding can cause significant latency and network congestion in this type of protocol.
- Hybrid protocols are combination of both proactive and reactive protocols. For example, in cluster-based routing, clusters are proactively formed and maintained in order to organize the network. Simultaneously, route discovery among clusters occurs reactively on demand, thereby reducing unnecessary control messages and conserving energy.

2.12.2 Structure-based routing protocols

Based on network structure, routing protocols can be flat-based, hierarchical-based or location-based.

- In flat-based protocol, the sink queries a particular set of SNs and then waits for a response. This protocol uses a data-centric routing strategy and assigns equal responsibility to each node.
- When using hierarchical routing, a cluster is created. The CHs are nodes with higher energy, while the low-energy nodes sense and send data to their associated CHs.
- The location-based routing protocol makes use of the nodes' location data to perform the routing operation.

2.12.3 Protocol operation-based

Based on protocol operation, routing is classifed as: multipath-based, query-based, negotiation-based, QoS-based and coherent-based or non-coherent-based

- Multipath selection is a method employed in multipath protocols to reduce communication delay and uniformly distribute the communication load among sensor nodes, hence improving network lifetime and reliability. Nevertheless, this approach has a significant energy consumption because it requires sending messages on a regular basis to maintain network routes.
- In a query-based protocol, a node generates a query that is disseminated throughout the network. The node(s) holding the requested data respond by transmitting it back to the querying node.
- Negotiation based protocol uses negotiation to reduce redundant data transmission.
- In a QoS-based routing protocol, three factors are considered while making routing decisions: the residual energy of the nodes, the path's quality of service, and the packet's priority.
- When using coherent routing, nodes forward data to the aggregator following minimal processing, which includes tasks like time-sampling and redundancy removal.

- In non-coherent routing, the nodes process the raw data locally before sending it to the aggregator for additional processing.

2.12.4 Next hop selection-based

Based on next hop selection-based, routing protocols can be categorized into broadcast-based, location-based, hierarchy-based, content-based and probability-based.

- Each node in a broadcast-based protocol determines whether or not to forward the message. If it chooses to forward, the message will just be rebroadcast; if not, it will be dropped.
- When using location-based routing, the next hop on the path to the destination is determined based on the known locations of the neighboring nodes and also the destination.
- All nodes forward their data to the aggregator in a hierarchical routing configuration. Aggregation of the data might lead to a reduction in communication overhead, which will ultimately result in less energy being used.
- A content-based routing protocol only considers the query's content when determining the next hop. Because the sink requests data only, regardless of where it comes from, this routing technique is compatible with WSN architecture.
- Routing protocols that rely on probability select the next hop according to computed probabilities. In this approach, the probability used to pick and maintain a set of paths is based, for example, on how much energy can be saved along each way.

2.13 Application domains of WSNs

WSNs are being used in a wide range of application fields. Future common things are expected to be equipped with sensors to become intelligent. This section presents the primary application domains of WSNs according to the taxonomy outlined in reference [1], illustrated in Figure 2.8. The details of each category are outlined below.

- **Military applications:** WSNs are widely used in the military field. They are utilised for a variety of purposes, such as tracking moving vehicles or soldiers while on a surveillance

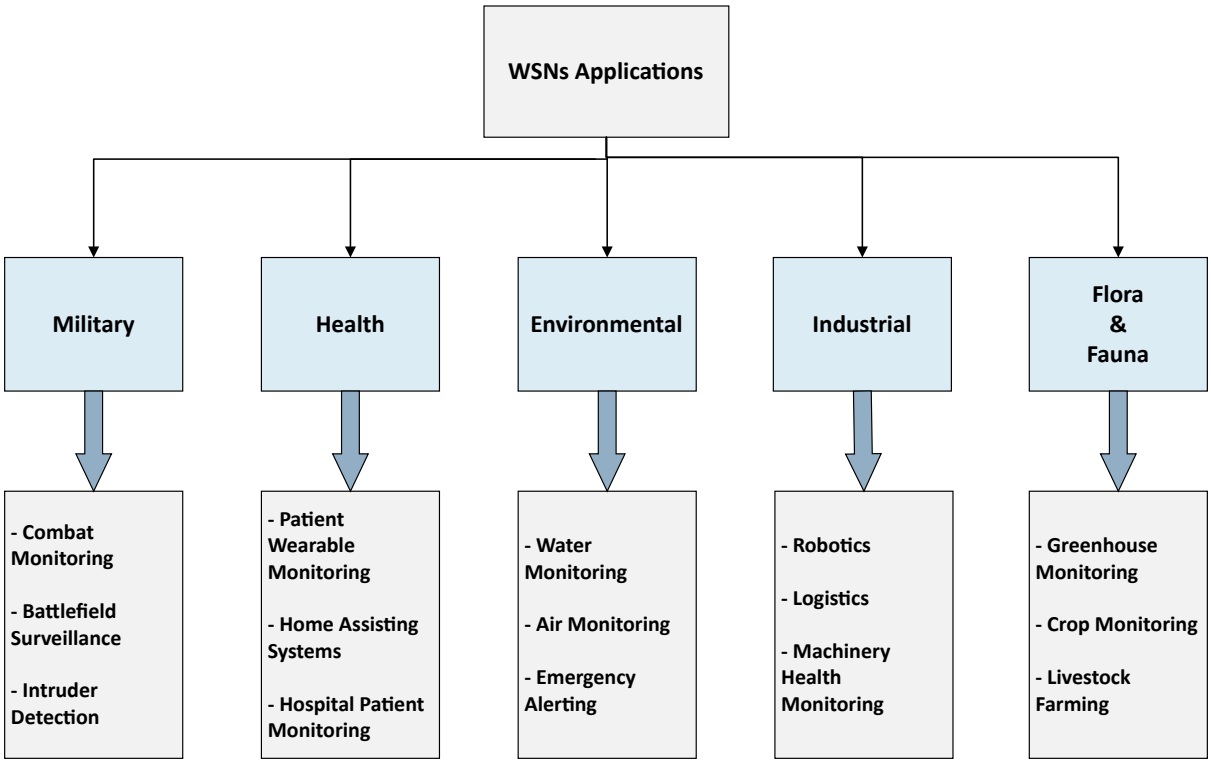


Figure 2.8: Application domains of WSNs [1].

mission. Additionally, they are utilised to gather information or data from the target region and transmit it to the sink. WSNs have been used in this military field for a very long time. The Sound Surveillance System (SOSUS), a military application that makes use of WSNs, was originally implemented by the US military in 1960. Submarines are detected and tracked using this method [22].

- **Health applications:** In the healthcare industry, WSNs utilize advanced medical sensors to monitor patients in hospitals and homes. They also use wearable devices to monitor patients vital signs in real time. The three primary subcategories for WSN health applications are wearable patient monitoring, home assistance systems, and hospital patient monitoring [23].

- **Environmental applications:** WSNs enhance environmental applications requiring continuous monitoring in hostile and isolated environments, including water, air, and emergency alerting. Various sorts of sensors, depending on the monitored area, are typically employed for this surveillance [23].

- **Industrial applications:** WSNs are useful in industry for keeping an eye on the state of industrial equipment or the manufacturing process itself. For instance, sensors can be

used by oil refiners or chemical factories to keep an eye on the state of their miles of pipes. These sensors are meant to notify users in the event that a breakdown occurs [24].

- **Flora and Fauna applications:** Every nation needs designated regions for the management and conservation of its own flora and fauna. WSNs offer a variety of applications designed to successfully monitor and support specific agricultural ecosystems. The principal subcategories of such applications include crop monitoring, livestock farming, and greenhouse monitoring [1].

2.14 Conclusion

In conclusion, WSNs are powerful tools for monitoring and interacting with the environment. They consist of SNs with specific functions and can be deployed in different ways. This chapter discussed the various functions of SNs, the sink role, and deployment scenarios for SNs. This chapter also addressed the architecture of WSNs and their communication schemes. WSNs face challenges like energy efficiency and communication constraints, but solutions like energy conservation methods and routing protocols help overcome these issues. With applications in military, healthcare, environment, industry, and wildlife, WSNs play a key role in many fields. Despite their challenges, WSNs continue to grow and provide valuable solutions for real-world problems.

Following this fundamental review of WSNs, we shall move on to the specifics about the utilization of MSs in data collection within these networks. This will be the primary focus of the subsequent chapter.

Chapter 3

Data Collection in WSNs via Mobile Sinks

3.1 Introduction

Mobile Wireless Sensor Networks (MWSNs) are an advanced type of WSNs that improve data collection and network efficiency by incorporating mobility into SNs, sinks, or both. Among the strategies being utilized, sink mobility emerges as a pivotal method for enhancing data collection efficiency and minimizing energy consumption. This chapter concentrates solely on data gathering with MSs. It initially explores the diverse forms of mobility common to MWSNs and describes the classification of these networks. It subsequently examines the advantages offered by MSs, their various mobility patterns, and the challenges they present. In the end, it presents a summary of pertinent literature, assessing previous research contributions and emphasizing current challenges to establish a foundation for future development.

3.2 Definition of MWSNs

An MWSN is composed of MS and/or mobile SNs that are able to move across the network. Mobility allows these nodes to change their position after being deployed [25]. An MWSN has more energy efficiency, better coverage, better target tracking, and more channel capacity than a static WSN [26].

3.3 Mobility forms in MWSNs

Three primary forms of mobility can be found in MWSNs [27]: SN mobility, event mobility, and sink mobility.

3.3.1 SNs mobility

In such a form of mobility, the network must constantly reconfigure itself to function properly in the context of SNs mobility. Due to the energy cost of geolocation techniques and their increased demand, the issue of node energy conservation becomes more challenging.

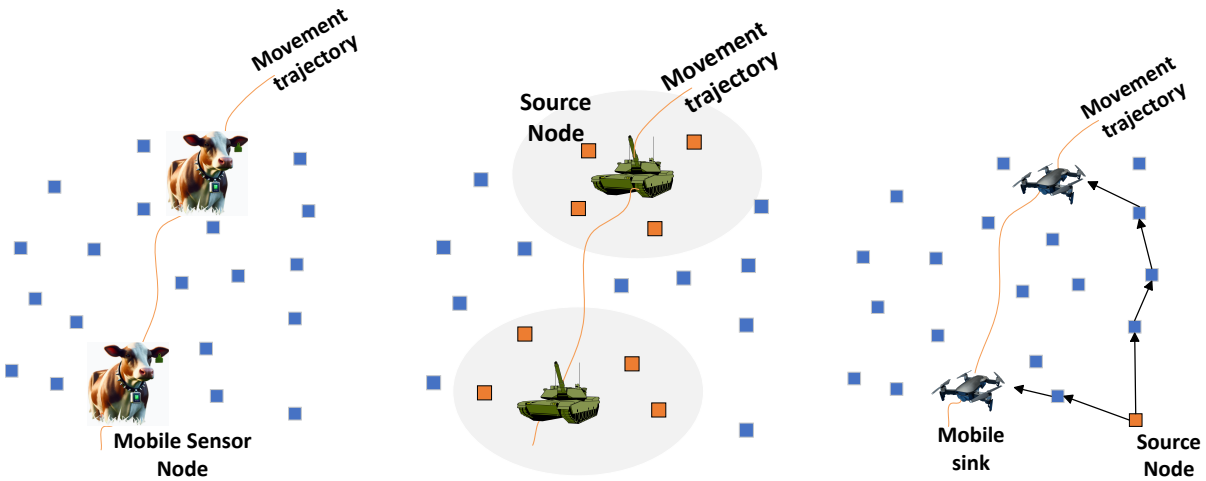
3.3.2 Event mobility

It is crucial that a sufficient number of nodes cover the observed event in this type of mobility. Consequently, when the target moves away from the SNs, they will go into sleep mode, while those that are close to the object will wake up and begin to actively monitor it.

3.3.3 Sink mobilty

Sink mobility occurs when the collection point is moveable. In this case, the SNs can wait for the sink's arrival to transmit their data directly or over fewer hops, hence minimizing energy consumption and network traffic, resulting in extended battery life.

The figure below (Figure 3.1) illustrates the different forms of mobility.



(a) SNs mobility (b) Event mobility (c) Sink mobility

Figure 3.1: Different forms of mobility in MWSNs [28].

3.4 Classification of MWSNs

MWSNs can be classified based on the mobility of sink nodes and ordinary SNs. Following are the key categories that can be identified in this classification [29]:

- **Sink and ordinary nodes are mobile:** This type of MWSN is designed for environments where sink and SNs must remain mobile. In these networks, external factors like as obstacles, weather, or mobility patterns may significantly impact their movement, connection, and performance.
- **Sink nodes are static and ordinary nodes are mobile:** Due to the mobility of SNs, this type of MWSN may effectively cover large areas without requiring a dense network of stationary sensors, making it appropriate for monitoring large regions efficiently.
- **Sink nodes are mobile and ordinary nodes are static:** This type of MWSN is popular for data gathering across large or dispersed areas. MS nodes collect data, while static SNs observe and send data. The sink node controls data processing, transmission, route maintenance, and energy usage, reducing the load on regular SNs.

3.5 Real life applications

In real life, MWSNs are used in a variety of fields. The most well-known uses are:

- **Military applications:** MWSNs are integral to contemporary military operations, particularly in surveillance and threat identification. A noteworthy example is the Wide Area Tracking System (WATS), a project created by the National Laboratory of Lawrence Livermore for the detection of nuclear weapons [30]. Sensors in this system increase the detection rate and decrease false alarms, making deployment simpler [27].
- **Medical applications:** MWSNs are increasingly utilized in healthcare for real-time monitoring and diagnosis. A novel platform for cardiac patient monitoring that uses sensors to gather information from a mobile phone and an ECG electrocardiogram, helping healthcare providers identify heart diseases. [27, 31].
- **Sport applications:** MWSNs are eventually employed in the sports sector for real-time player surveillance, trajectory monitoring, and performance evaluation through wearable

sensors and mobile data gathering devices. [27, 31].

- **Environmental applications:** The deploying of MWSNs significantly enhances environmental monitoring in remote and difficult to reach places. A proposal to build an MWSN on the Reventador volcano in Ecuador's western Amazon aims to track ocean water temperature, salinity, and velocity using sensors and satellite data [30, 31].

- **Ecological habitat monitoring applications:** MWSNs have emerged as vital tools for monitoring wildlife and examining biological ecosystems in isolated regions. The Zebranet system from Princeton University, which is situated in Kenya, tracks zebra populations using sensors that are affixed to the animals. With the use of wireless transducers, GPS, CPUs, and flash memory necklaces, the system seeks to give researchers a large amount of data over time [27, 32].

- **Ambient Assisted Living applications:** The purpose of this approach is to help the elderly live better, more independent lives by the utilization of mobile sensors, either worn on the body or situated within their surroundings. These sensors track vital signs and daily activities, transmitting data wirelessly to caretakers or medical services [27, 33, 34].

3.6 Mobile sink definition

A MS in MWSNs refers to a data collector that may move around the network to gather data from source nodes. This is done to enhance coverage and reduce the number of hops required to get to each node [35]. In real-life applications, MSs can take various forms, including Unmanned Aerial Vehicles for large or remote areas, Unmanned Ground Vehicles (UGVs) for urban or industrial settings, Unmanned Maritime Vehicles (UMVs) for aquatic environments, ground robots for structured spaces, and stratospheric balloons for wide-area monitoring. The form of the MS is chosen based on the environment and application requirements, making it a flexible and efficient solution for data collection [36].

3.7 Mobile sink benefits

MSs offer significant advantages in MWSNs, such as [2]:

- **Hotspot mitigation:** The hotspot problem is a prevalent challenge encountered by WSNs with static sinks. The hotspot issue arises when nodes in proximity to the sink deplete their energy at a faster rate than other nodes. This results from the concentration of data traffic directed towards this static sink. A MS mitigates this problem by collecting data from SNs while navigating the network. The nodes adjacent to the sink are not consistently identical, hence preventing the continuous burden on the same nodes.

- **Load balancing:** Dynamic changes in the sink's neighboring nodes distribute network load evenly.

- **Shorter paths:** Reduced data dissemination paths increase throughput and decrease energy consumption.

- **Improved network handling:** MSs enhance performance in managing network connectivity and efficiency.

3.8 Sink mobility patterns

A MS may use a variety of mobility patterns to move through the area of interest. These patterns are categorized into predictable/fixed-path mobility patterns, random mobility patterns, and controlled mobility patterns [27, 37]: .

1. **Predictable/Fixed-Path mobility pattern:** In this pattern, the MS moves within the sensing field's perimeter, following a *predetermined path*, and pauses at certain positions [2, 37].

2. **Random mobility pattern:** In this pattern, the MS moves *randomly* within the sensing field's perimeter, while pausing at various positions [37].

3. **Controlled mobility pattern:** In this pattern, the MS's movement is determined and controlled by the observer, who may be an operator who continually alters the sink's trajectory in accordance with network conditions or application requirements [2, 38].

3.9 Challenges in deploying mobile sinks

The use of MSs constitutes a promising solution for data collection across various locations. However, their deployment faces many challenges, among them:

1. **The detection of sink contact:** In order to establish a connection with a MS, the SNs must first identify if the sink is within their communication range. However, a moving sink may remain undetected by the nodes that are in sleep mode [38].

2. **Packet loss ratio increase:** Each SN forwards data to the last known position of a MS, but effective transmission is compromised if the most recent mobility information is not sent across the network. When a MS deviates significantly from the last known sojourn point, the issue becomes more complex, leading to message drop due to lengthy traversal time [38].

3. **The MS's energy consumption:** The MS is powered by a battery. In order to save energy and prolong the sink's operating life, its traveling distance must be reduced [39].

4. **Mobility-aware duty cycle management:** In order for SNs to transmit topological updates to a sink, they must be in listening mode. The nodes might improve sink identification if the sink's mobility could be anticipated or calculated by taking advantage of the knowledge about the sink mobility pattern [38].

5. **Identification of the optimal sink sojourn locations:** The MS in MWSNs must identify the best places (sojourn locations) to pause in order to gather information from SNs. When the communication mode is periodic, that is, the sink gathers data at regular intervals, this becomes very difficult [40].

6. **Determination of the optimal trajectory of the sink node:** The MS must follow the most efficient route to reach the sojourn locations after they have been established. The trajectory should ensure that each node is efficiently covered while also reducing the total distance traveled [2].

7. **Obstacle avoidance:** In real-world deployments, the sink's movement may be hindered by physical obstacles (such as buildings, terrain, or other barriers). The sink must navigate around these obstacles while still maintaining an efficient trajectory [2].

8. **Higher message delivery delay in fixed path mobility pattern:** Delays in message transmission may occur when the sink travels a fixed path (such as a preset route), particularly if the path is lengthy or the sink is traveling slowly. This may result in a higher data-collecting delay [2].

3.10 Related works

This section outlines various techniques that have been developed thus far to identify the best sojourn places for the MS, all of which rely on a pre-defined mobility pattern.

In [4], the Energy- and Delay-Efficient Data Acquisition (EDEDA) protocol for WSNs was proposed. In this approach, the sensor field is partitioned into a virtual grid structure as depicted in Figure 3.2, with each grid cell containing a certain set of SNs, one of which is designated as the Grid Cell Head (GCH). Other cell nodes supply information to these GCHs. To gather data from adjacent GCHs in a single-hop manner, the MS pauses at specific grid cells designated as Visiting Cells (VCs). Before returning to the base station to transmit data, the MS traverses a predetermined Hamiltonian cycle that passes through all VCs. This design minimizes energy usage and decreases the travel distance of the sink, hence reducing delay in data collection. EDEDA incorporates a method for substituting GCHs upon depletion of their energy to maintain balanced energy consumption and prolong network longevity.

The authors in [41] proposed a new virtual grid-based rendezvous point and sojourn location selection method, termed VGRSS. A virtual grid is used to represent the network, with specific nodes designated as rendezvous points (RPs). These RPs collect data from nearby sensor nodes and forward it to a MS. To select optimal RPs, the method employs fuzzy logic, considering each node's remaining energy and its central position within its grid cell. This approach helps prevent individual nodes from becoming overloaded or depleting their energy too quickly. Rather than stopping at every RP, the MS only pauses at the intersections of four grid cells (See Figure 3.3), referred to as sojourn locations. This strategy reduces the sink's travel distance, thereby accelerating the data collection process. The grid is initialized once, and only the RPs are updated as necessary, eliminating the need for frequent reconfiguration of the entire network.

The same authors proposed in [42] an alternative approach to the identical data collection

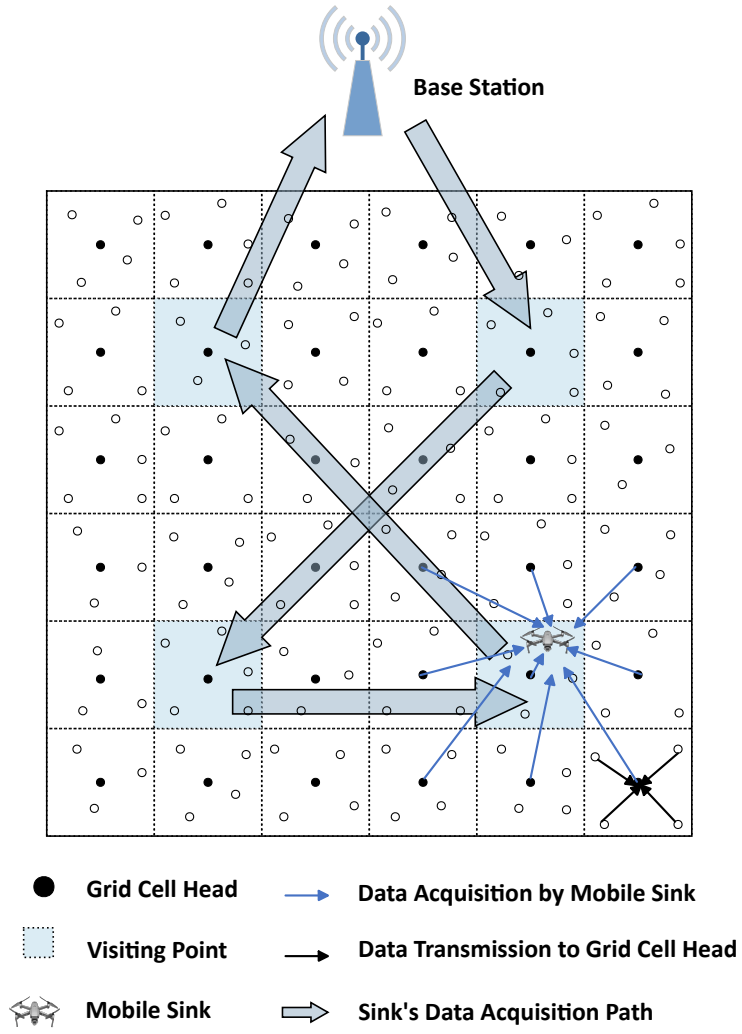


Figure 3.2: EDEDA's principle [4].

problem using MS. They considered a virtual grid structure to plan the sink's trajectory and alter the data transmission routes towards the grid cell currently occupied by the MS. A $K \times K$ virtual grid is created according to the number of deployed sensor nodes, where K is an even integer. Within each cell, a GCH is chosen based on the proximity of the GCMs to the cell's center and their remaining energy. Another GCH within the same cell is re-elected periodically to avert the depletion of the battery of the currently elected GCH. The sink's trajectory within the sensor network is designed to navigate the boundaries and center of the sensing field. Four paths are specifically examined: the two lines, 1 and K, of the grid, in addition to the two diagonals of the grid. The MS randomly selects a cell identified as a Rendezvous Grid Cell (RGC) from those within each path to collect data from the corresponding GCHs. When the MS remains at a certain grid cell, it notifies the GCH of the last recorded Rendezvous GCH (RGCH) of its current position using a specified packet. This RGCH initiates the transmission of this packet

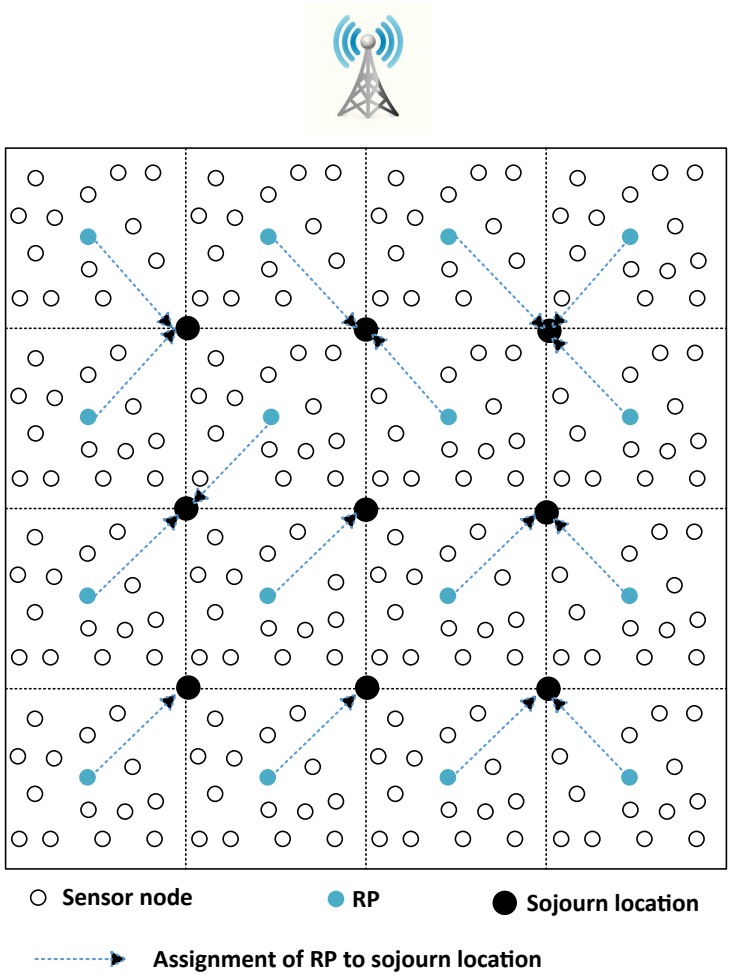
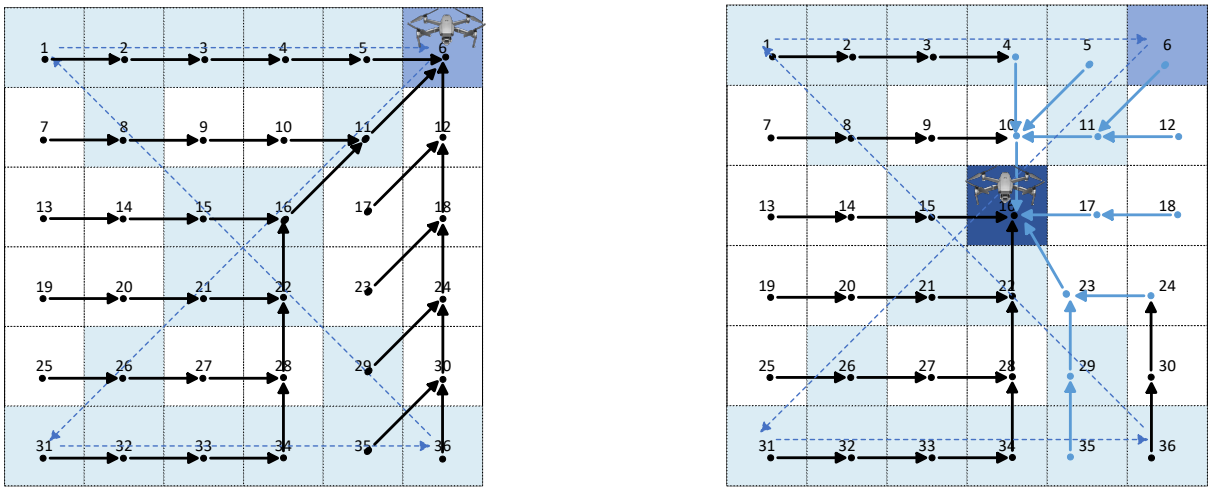


Figure 3.3: VGRSS's principle [41].

to all GCHs within the network. This transmission enables the creation or modification of data routing paths to the RGCH, which thereafter forwards all received data to the MS. Routing paths are established by evaluating the minimal hop count linking each GCM to the RGCH. Figure 3.4 illustrates the rationale behind TARA.

In [40], the Energy Efficient Optimal Sojourn Location of MS (E2OSLMS) method was proposed to optimizes the sojourn locations and traveling path of MSs. As shown in Figure 3.5, The authors presumed a circular sensing region, which they partitioned into a specified set of coronas. Each corona encompasses a specified number of clusters. A cluster head is selected within each cluster according to the Euclidean distance to the cluster's center and the residual energy of each candidate. Four MS are considered that traverse a spiral trajectory originating from the center of the monitoring area, traveling through a specified set of designated clusters, and subsequently returning to the initial point. The method selects the optimal sojourn locations using a fuzzy inference system (FIS). The location of the MS, where it must reside within each



(a) (b)

Figure 3.4: Data gathering and TARA route adjustment while sink (a) is at grid cell 6 and sink (b) travels from grid cell 6 to 16 [42].

cluster, is determined from various member node positions. The FIS evaluates nodes based on three criteria: residual energy, distance from the MS, and a preset weight factor. Priority for sojourn points is given to nodes with lower weight values, more remaining energy, and closer proximity to the sink. Additionally, the protocol balances energy consumption across nodes and cluster headers by dynamically modifying the MS's sojourn duration inside each cluster.

In [43], the goal of the Energy-Aware Path Construction (EAPC) is to establish an energy-efficient data-gathering path while enhancing the selection of data collection points (CPs), under the constraint of a predetermined path length provided as input. The algorithm comprises three primary phases: initialization, selection of CPs, and path construction. A minimal spanning tree (MST) is established during the initialization phase to minimize the total path length. The collection point selection phase determines CPs using a benefit index that optimizes energy usage while minimizing the forwarding burden on sensor nodes. The most direct route between the BS and the CPs is finally generated during the path construction phase. EAPC's principle is illustrated in Figure 3.6.

In [44], a novel energy-efficient data collection algorithm is proposed, termed BIIE, to balance energy usage of sensor nodes. This algorithm operates through three principal phases: initialization, path formation, and local re-clustering. During the starting phase, sensor nodes are grouped utilizing an advanced hierarchical clustering method that minimizes communication expenses while taking the route restrictions of the MS into account. To provide a balanced

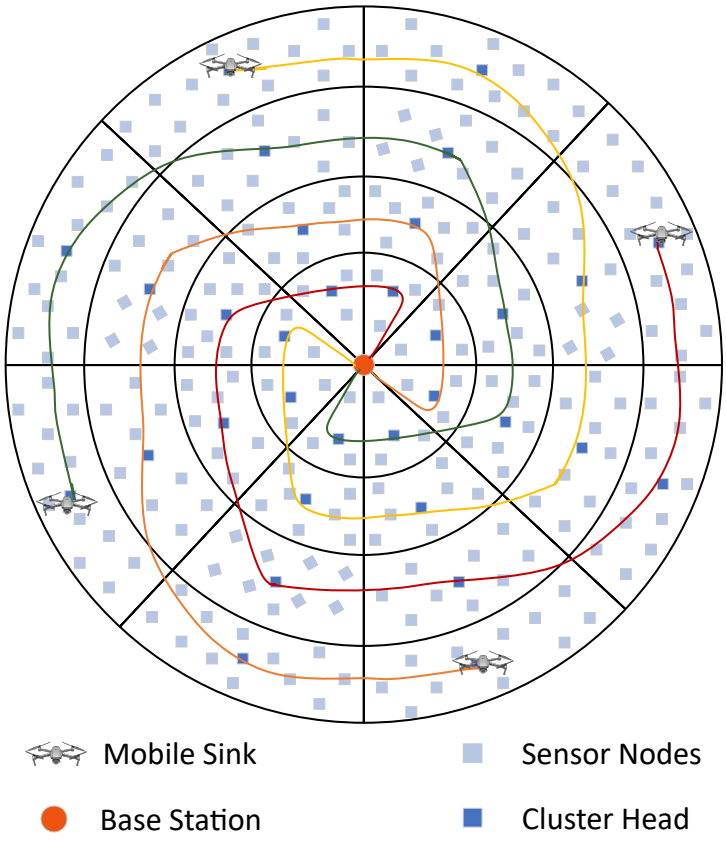


Figure 3.5: E2OSLMS's principle.

inter-cluster energy distribution and establish an efficient traveling path for the MS, Particle Swarm Optimization (PSO) is employed to select the optimal rendezvous nodes (RNs) within clusters during the path design phase. To mitigate the load on low-energy nodes and sustain energy balance within clusters, the local reclustering phase adaptively alters clusters based on the residual energy levels of sensor nodes. The principle of BIIE is somewhat analogous to that of EAPC.

In [45], the Hybrid Rendezvous Clustering Model (HRCM) was designed to improve data collection in WSNs. It combines the strengths of rendezvous-based and cluster-based approaches to reduce energy consumption and minimize delays. Using the LEECH algorithm, clusters are formed, and cluster heads are chosen based on factors as node density, residual energy, and closeness to MS paths. Key collection points, known as Steiner points, are identified and grouped with Particle Swarm Optimization (PSO) to plan efficient routes for MSs. These sinks then follow optimized paths, calculated using Dijkstra's algorithm, to collect data (see Figure 3.7). The protocol also decides whether to store or forward data dynamically, depending on packet deadlines and sink arrival predictions.

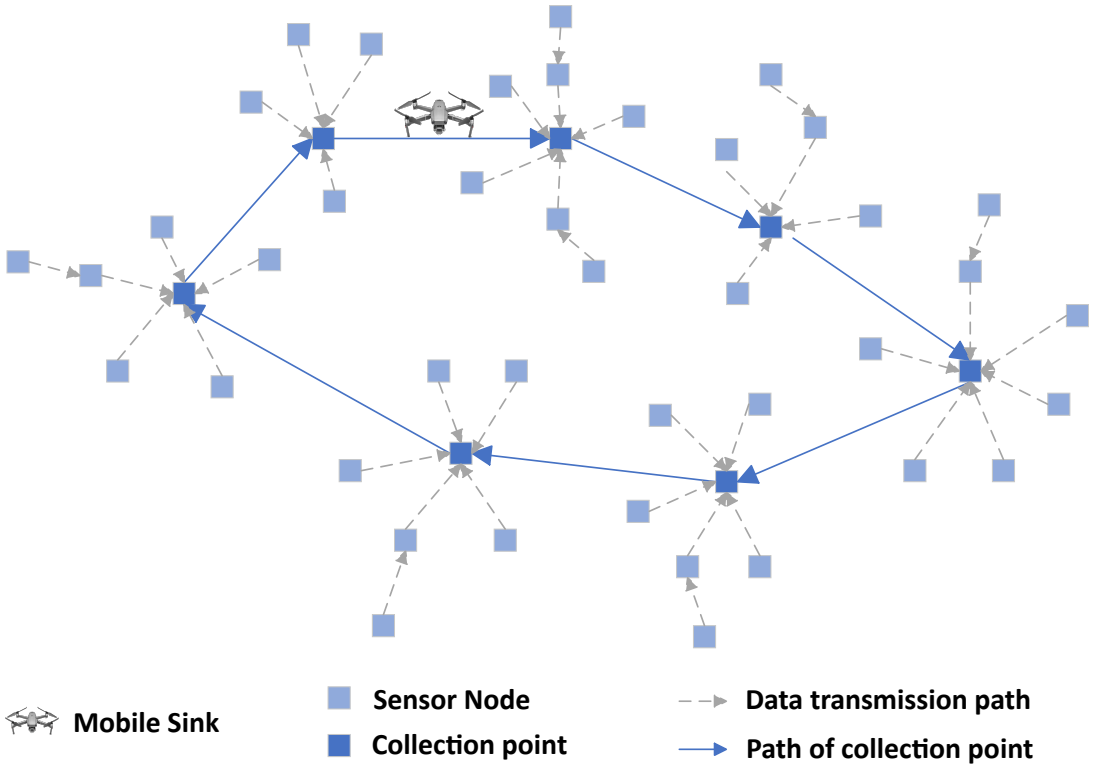


Figure 3.6: EAPC's principle.

An optimal RPs selection with MS trajectory construction algorithm is proposed in [46]. This approach, called ORPSTC, determined the optimal RPs for data collection using a clustering technique based on Minimum Spanning Trees (MST). It also used computational geometry to create an efficient path for the MS, reducing data latency, balancing energy consumption across SNs, and extending the network's lifetime. To enhance its performance and flexibility, ORPSTC included virtual rendezvous points (VRPs) and allowed for dynamic re-selection of RPs. The various steps of ORPSTC are depicted in Figure 3.8.

In [47], the authors presented an enhanced approach to identify the optimal path for MS with clustering techniques for both homogeneous and heterogeneous WSNs. They considered both scenarios — single and multiple MSs. This proposed method divides the sensing field into several equally sized regions based on the required number of deployed MSs. A proposed clustering method called SEA, which stands for Stable Election Algorithm, organizes the sensor nodes into clusters. SEA is designed to reduce the frequent rotation of the CH role to avoid the continual transmission of control messages for CH reelection. Initially, sensor nodes with the highest number of neighboring nodes are selected as CHs in a homogeneous WSN. In return, the residual energy of nodes is considered to prioritize super-nodes with abundant energy resources as CHs in the case of a heterogeneous WSN. The selected CHs report their

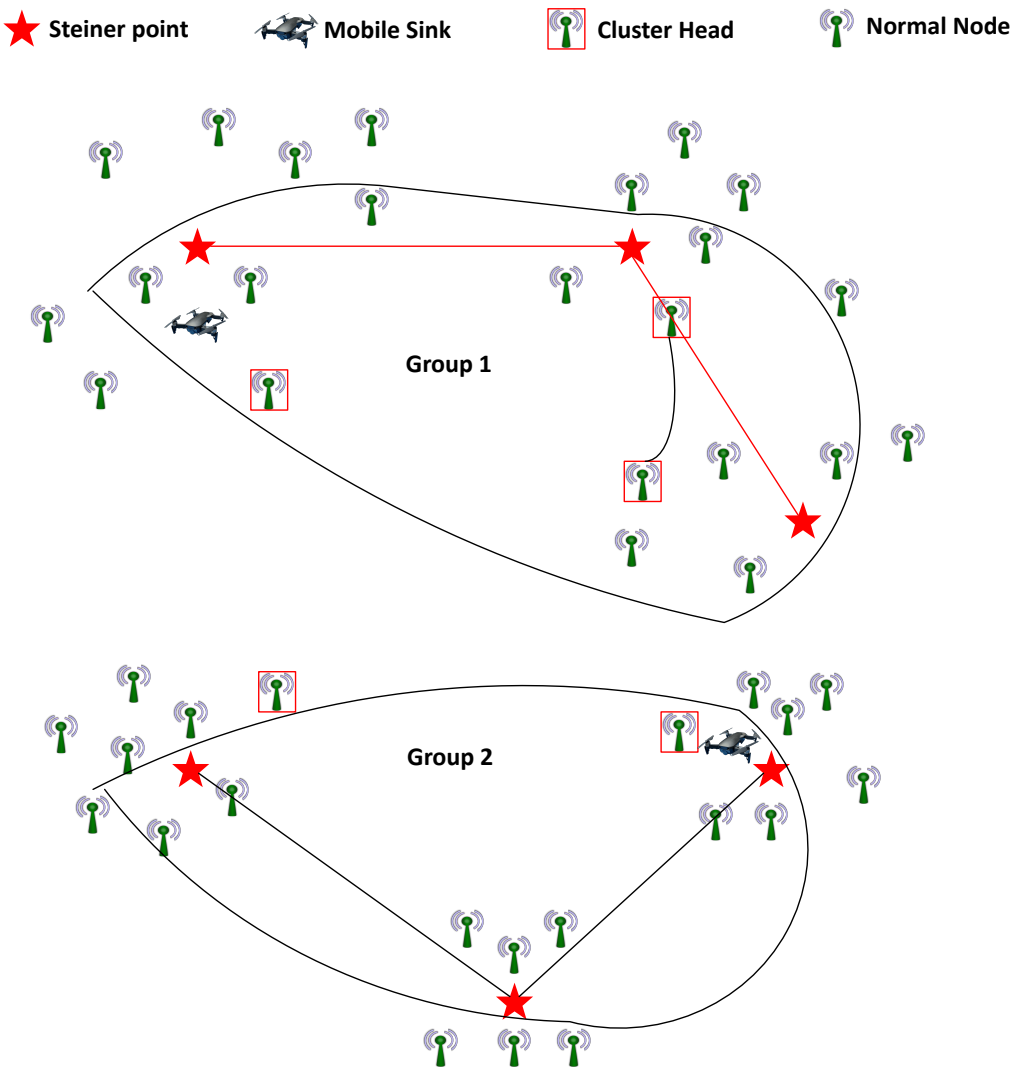


Figure 3.7: HRCM's principle [45].

positions to a central base station to compute the path of the MS. The determination of the VPs to visit is formulated as a minimum weighted vertex cover problem (MWVCP) and a proposed sojourn method is developed to address it. The path of the MS is then computed using multi-objective evolutionary algorithms where the following three of them are considered: ant colony optimization (ACO), genetic algorithm(GA), and simulated annealing (SA).

The authors in [48] outline the following method to ascertain a minimum number of RPs. The WSN was represented as an undirected graph G. They subsequently determined the connected components of G through the application of Breadth-First Search (BFS) traversal. The reasoning for this is to diminish the problem's size, hence decreasing the computing complexity of the solution. The larger connected components are then partitioned into manageable units (MUs) using the combined use of a heuristic and Tarjan's technique. Upon acquisition of these

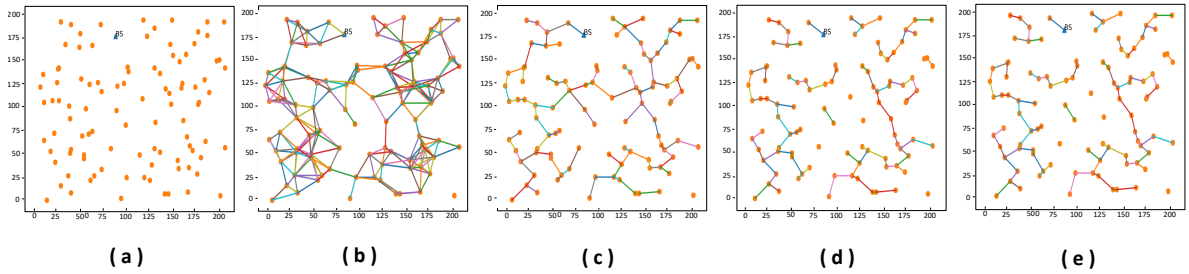


Figure 3.8: MST-based clustering and RPs selection: a) Initial deployment of 100 SNs; b) A WSN as a graph; c) Minimum-cost spanning tree; d) MST-based clusters; e) RPs selection [47].

MUs, spectral clustering is employed to create clusters from the MUs. The centroid of each cluster is then identified as RP. The identified RPs are ultimately employed to construct the traveling salesman tour (TSP) for the MS utilizing Christofides' algorithm.

3.11 Discussion

We have previously examined a set of recent articles related to our work. Table 3.1 outlines the summary and key features of the examined set of works. This review aimed to explore how these studies manage to reduce energy consumption, identify the MS's visiting points and plan its trajectory, reduce the data collection delay, and also set up nodes so that the design goals can be met. In the following, we list the main similarities, differences, and compromises between these examined strategies according to a set of key characteristics:

1. **Sink trajectory pattern:** All methods employ predetermined sink trajectories[1] where the sink follows a predefined path to collect data.

2. **Clusterization:** is used in all methods to improve scalability and energy efficiency. SNs are organized into clusters, and data is aggregated at RPs or CHs to save energy and cut down on redundant transmissions.

3. **Structural differences:** Different structural approaches affect energy consumption and latency:

    - **Tree structures:** Techniques such as EAPC and BIIE employ tree-based data forwarding, wherein SNs send data to RNs or VPs over a number of hops without waiting for the MS's arrival. This reduces data latency but increases energy con-

[1]Please refer to Section 3.8 for more details regarding this trajectory pattern.

sumption due to the frequent transmission of data over multiple hops.

- **Grid structures:** VGRSS and EDEDA are grid-based methods that minimize energy consumption by organizing SNs into a structured grid as well as reducing the number of data transmissions through one-hop data collection. Heuristic techniques are employed to optimize the path of the MS, hence reducing data gathering latency.

- **Hybrid structures:** TARA uses a hybrid structure that combines tree and grid components. The sensing field is first partitioned into a finite number of grid cells, followed by the construction of a tree that connects the multiple GCHs to the RGCH. This tree is continuously updated according to the MS's current location to keep providing short-hop paths between each GCH and the RGCH. This method improves data forwarding efficiency; however, it also increases energy consumption due to the frequent tree updates necessitating an important exchange of control messages.

4. **Single vs. multi-sink deployment:** The examined approaches differ in whether they utilize one or multiple MSs for data collection, a factor that directly impacts their design decisions.

5. **Variability in energy consumption and data acquisition latency:** The many studied data collection methods reveal different levels of energy consumption and latency, determined by their design decisions and operational constraints:

   - **High energy, low latency:** Methods such as EAPC, TARA, BIIE, and HRCM provide low data latency. However, they are marked by significant energy consumption, which may result from the frequent modifications of the implemented routing structures necessitating substantial control overhead or from the extensive data transmissions required to relay data to the MS. Particularly:

     – **EAPC and BIIE:** The determination of the MS's trajectory is constrained by the predefined traveling distance given as input. If a short path is used, fewer RNs are selected, reducing latency. Nevertheless, data transmission across the established trees linking non-RNs to RNs incurs significant energy expenditure, as nodes are required to relay both their own data and that of others.

  - **TARA:** As previously explained, this approach maintains an updated shortest-hop routing tree that connects each GCH to the current RGCH. Instead of awaiting the arrival of the MS, GCHs transmit their data directly from node to node. This results in reduced latency; yet, a significant amount of energy is used due to the frequent exchange of control messages required for tree maintenance.

  - **HRCM:** Conserves energy by using rendezvous-based collecting awaiting the MS's arrival before initiating data transmission; but, when quick delivery is required, it shifts to cluster-based forwarding, which uses more energy.

- **Low energy, medium latency:** Methods such as E2OSLMS, EDEDA, SEA, SCTR, and VGRSS achieve low energy consumption for two primary reasons. First, they rely on one-hop data collection, wherein SNs await the MS's arrival to communicate their data immediately, hence eliminating any need for multi-hop forwarding. Secondly, in contrast to EAPC, TARA, and BIIE, these solutions do not necessitate regular up keep of routing structures between SNs and the MS, thus reducing control overhead. Regarding data collection delay:

  - The EDEDA, SEA, SCTR, and VGRSS approaches all aim at minimizing the number of MS's visiting points to shorten its path, thereby reducing data gathering latency. Nevertheless, as SNs must await the arrival of the MS prior to delivering their data, the overall delivery latency remains moderate.

  - E2OSLMS: Four MSs are used to gather data from SNs. Nevertheless, each MS traverses each assigned cluster in its established route, leading to a considerable number of VPs being visited, hence resulting in significant data gathering latency in comparison to the previously cited approaches. Nonetheless, the availability of several MSs mitigates this delay. Taking both considerations into account, we categorize the latency as moderate.

- **High energy, medium latency:** ORPSTC uses multi-hop forwarding, increasing energy usage but achieving only medium latency since data travels through a few number of SNs before reaching the sink.

- **Hybrid Approaches:** HRCM balances energy and latency by switching between rendezvous-based (low energy, high latency) and cluster-based forwarding (higher energy, lower latency), depending on network conditions. (Adaptive Trade-off)

6. **Path planning strategies:** The latency and energy consumption of data gathering are greatly impacted by the MS movement throughout the network. The planning methods that the different examined approaches deal with are as follows:

   - **Computation-free path strategy:** Methods like E2OSLMS and TARA use predefined paths for the MS, which don't require real-time computation. Although this lowers the computational overhead, it does not always yield the most optimal trajectory in terms of minimizing latency or energy consumption.

   - **Optimization algorithms:** Most approaches rely on well-established optimization methods to determine the most effective routes for the MS. BIIE uses particle swarm optimization, HRCM employs Dijkstra's shortest path algorithm, VGRSS applies the Christofides algorithm, etc. Although these methods can adjust to network changes, they might need more processing power. Alongside these conventional strategies, certain authors have proposed their own planning methods. Significantly, EAPC and ORPSTC are the sole techniques among those analyzed in which the authors have devised a custom planning method.

In summary, the examined solutions optimize energy cost and data collecting delay in different ways. Specifically, the following observations can be highlighted:

- Techniques where data collection latency is not critical don't involve significant energy consumption as they primarily depend on single-hop data transmission without necessitating multi-hop forwarding. Furthermore, they do not primarily establish or maintain routing structures to transmit data from SNs to the MS, hence eliminating the control overhead required for this operation.

- However, in latency-sensitive scenarios, increased energy consumption arises from the significant control overhead resulting from this continuous upkeep of routing structures, alongside multi-hop data forwarding, wherein SNs transmit their data directly to the MS without waiting for its arrival.

Table 3.1: Summary of the reviewed articles.

Method	Structure Type	Sink Trajectory Pattern	Multi-Sink Support	Path Selection Strategy	Energy Consumption	Data Acquisition Latency
E2OSLMS [40]	Ring	Predetermined	Yes	Computation-Free Path Strategy	Low	Medium
EAPC [43]	Cluster	Predetermined	No	Convex polygon algorithm	High	Low
TARA [42]	Hybrid (Grid & Tree)	Predetermined	No	Computation-Free Path Strategy	high	Low
EDEDA [4]	Grid	Predetermined	No	Hamiltonian Path Planning Strategy	Low	Medium
BIIE [44]	Cluster	Predetermined	No	Particle Swarm Optimization (PSO)	High	Low
HRCM [45]	Cluster	Predetermined	Yes	Dijikstra shortest path algorithm	High	Low
ORPSTC [46]	Cluster	Predetermined	No	Author-Defined Computational geometry approach	High	Medium
Approach in [47]	Cluster	Predetermined	Yes	Multi-Objective Evolutionary Algorithm (MOEA)	Low	Medium
SCTR [48]	Cluster	Predetermined	No	Christofides Algorithm	Low	Medium
VGRSS [41]	Grid	Predetermined	No	Christofides Algorithm	Low	Medium

3.12 Conclusion

In this chapter, we explored different mobility forms and the challenges of using MSs for data collection in WSNs. When used effectively, sink mobility can improve data gathering, extend the network's lifetime, and make better use of energy. However, there are still obstacles to overcome, such as managing sink movement, optimizing routing, and dealing with deployment limitations. Additionally, we reviewed previous studies that tackle various facets of mobile sink-based data collection, recognizing their contributions and pointing out areas that require further development. In the next chapter, we will present our own approach, which aims to overcome these obstacles and further improve network performance.

Chapter 4

EDCVP: Enhanced Data Collection via Visiting Points

4.1 Introduction

This chapter explains in detail our proposed solution, named EDCVP. We begin by describing the network model we used, as well as the underlying assumptions that shaped our design. This is followed by a brief overview of our proposal, highlighting its foundational concepts. We then review the list of data structures that facilitate its operation and the format of messages utilized for communication between nodes. The remainder of the chapter is then devoted to an in-depth discussion of how the protocol functions, detailing each sequential step and mechanism involved. Through this, we illustrate the significant interactions between nodes, the network structure, trajectory planning for the MS, and the dynamic modifications integrated into the system.

4.2 Network model

We use a particular network model to organize our research on WSNs. The key components and characteristics of our network design are shown in this model, which is described below:

- One MS node, a BS situated outside the sensing region, and a collection of uniformly distributed SNs make up our network. It is presumed that every node can identify its

location within the deployment region. Several aspects of our proposed data collection method, including the MS's movement planning, depend on this location knowledge.

- Starting from the BS, the MS travels to different locations around the network to gather data. When its job is over, it returns to the BS to deliver the data it has gathered.

- The MS travels freely within the sensing area, as the environment is presumed to be devoid of physical obstacles.

- Communication between nodes in the network is only possible if their distance from one another is equal to or less than their communication range, denoted as R_{max}. A common method of representing this communication range is to draw a circle with each node at its center and its communication range as its radius. As a result, nodes can communicate with other nodes that are inside their communication circle, facilitating the effective spread of information across the network. Specifically, the network topology is represented by a graph $G = \{V, E\}$, where E is the set of links connecting the nodes in the network and $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices that represent its nodes. This set has the following definition:

$$E = \{(u, v) \mid u \in V, v \in V, \delta_{u,v} \leq R_{\max}\} \tag{4.1}$$

where

$$\delta_{u,v} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2} \tag{4.2}$$

represents the Euclidean distance between the two nodes u and v, given that $u = (x_u, y_u)$ and $v = (x_v, y_v)$.

4.3 Overview of the proposed approach

The proposed approach combines the idea of clustering with the mobility of the MS. Instead of requiring the MS to visit every node individually, we select VPs at strategic locations where the sink can efficiently gather data from nearby SNs. This structured approach aims to improve data collection efficiency through a carefully planned sequence of steps:

- **Virtual grid construction:** To organize the network, we divide the sensing area into a virtual grid of square cells. The communication range of SNs determines the size of each cell guaranteeing that nodes within two adjacent cells can communicate with one another in a single hop.

- **Election of grid cell heads (GCHs):**

  – Each cell functions as a cluster, with the GCH selected from the sensors of the specified cell. Before any subsequent transmission of data, the GCH must gather and aggregate the data collected from its cluster members.

  – The GCH is selected based on its proximity to the cell's center and its residual energy level. This ensures that the node with the highest energy and optimal position is chosen to minimize energy consumption during data aggregation and transmission.

- **Determination of rendezvous cells (RCs):**

  – RCs are designated inside the grid, where each RC is responsible for gathering data from its eight surrounding cells. The GCH of an RC, referred to as the rendezvous GCH or RGCH, aggregates data from these neighboring cells and transmits it to the MS.

  – The selection of RCs is based on a predefined formula that ensures efficient data aggregation and transmission.

- **Determination of VPs:**

  – The VPs are chosen at the intersections of four adjacent cells. Because the MS may collect data from four nearby RGCHs in a single hop, redundant data transfers are minimized by carefully selecting these points.

  – The VPs are determined using a formula that ensures optimal coverage of the grid while reducing the number of stops the MS needs to make.

- **MS's path planning:**

  – The MS travels an established route that stops at each VP precisely once, starting

and ending at the BS. This path is modeled as a Hamiltonian cycle, ensuring that the sink covers all VPs efficiently.

- A backtracking algorithm is used to calculate the path, which resolves the Hamiltonian cycle issue and guarantees that the MS's path is optimized for data collection and energy efficiency.

- **Data collection process:** The sink node uses Hamiltonian cycles to gather data. At each VP, it broadcasts a *DATA_REQUEST* message, which is forwarded to RGCHs, GCHs, and SNs. Each SN prepares a *DATA* message, which is sent back to the MS. This process continues until the MS returns to the BS.

- **Re-election of GCHs:**

  - The GCH's energy may run out faster than those of other regular nodes because of the extra responsibilities they play. Their replacement is required in this situation. The RGCHs and other GCHs evaluate their energy levels on a regular basis. If this level is below a specified threshold, a specific message initiates the re-election process.

  - The process for choosing a new GCH will be similar to how existing GCHs were first chosen.

4.4 Detailed description of our proposal

4.4.1 Data structures used

Data structures in our approach are categorized into three types: those used by SNs, those used by the MS, and those used by the BS.

4.4.1.1 Main data structures used at the SN level

The following data structures are used at each SN i:

- Id_i**:** Unique identifier of SN i.

- $IdGCH_i$**:** Identifier of the *GCH* associated with node i.

- $IdRGCH_i$**:** Identifier of the *RGCH* to which node i has been associated, provided that *i* is a GCH.
- $Coord_i$**:** Coordinates of SN *i* (x_i, y_i).
- $IdCell_i$**:** Identifier of the cell in which the node *i* is located.
- $CoordCell_i$**:** Coordinates of the cell (row and column number in the virtual grid).
- RE_i**:** Residual energy of SN *i*.
- *IdSink***:** Identifier of the *MS* to which the node *i* transmits its data.
- $IAmGCH_i$**:** Boolean flag indicating whether SN *i* is a *GCH*.
- $IAmRGCH_i$**:** Boolean flag indicating whether SN *i* is a *RGCH*.
- $CellNList_i$**:** List of neighboring nodes of *i* located within the same cell. Each item in this list corresponding to each neighbor *n*, comprises:
  - Id_n: Identifier of neighboring node *n*.
  - (x_n, y_n): Coordinates of node *n*.
  - RE_n: Residual energy of node *n*.

4.4.1.2 Main data structures used at the MS level

The data structures used at the MS node, in addition to its own identifier and coordinates, are:

- *IdBS***:** Identifier of the BS.
- *CoordBS***:** Coordinates of the BS.
- *ListVPs***:** The set of *VPs* that the MS follows for data collection.

4.4.1.3 Main data structures used at the BS level

A BS participates in determining *VPs* and planning the MS's trajectory. In addition to its own identifier and coordinates, the BS also uses:

- *IdSink*: Identifier of the MS.
- *ListVPs*: The identified set of *VPs*.

4.4.2 List and formats of messages used

The operation of our protocol relies on the exchange of various control messages between network nodes. Table 4.1 summarizes these messages along with their roles, followed by detailed descriptions in subsequent sections.

Table 4.1: List of messages used.

Message	Description
GCH_ELECTION	Used by nodes to trigger the election of a GCH within a cell.
GCH_ANNOUNCEMENT	Broadcast by the elected GCHs to inform cell members of their roles.
DATA_REQUEST	Sent by the MS upon reaching a VP to request data from adjacent RGCHs. It is also used by the RGCHs and GCHs to request data from their respective GCHs and member SNs.
DATA	Used to send the observed data.
GCH_REELECTION	Triggers re-election of a GCH when its energy drops below a threshold.

GCH_ELECTION message

Every SN in a cluster sends this message at the start of a GCH election or in response to a reelection message. Allowing nodes to learn about other cluster members and their characteristics is its main goal. It has the main following fields:

- *IdNode:* Unique identifier of the sending node.
- *Coordinates:* Position of the sender.
- *IdCell:* Identifier of the cell to which the sender belongs.
- *RE:* Remaining energy of the sender.

GCH_ANNOUNCEMENT message

To announce its responsibility to its cluster members, a newly elected GCH broadcasts this message. It has the following fields:

- *IdGCH:* Identifier of the elected GCH.
- *Coordinates:* Position of the GCH.
- *IdCell:* Identifier of the cell to which the GCH belongs.
- *IAmRGCH:* Boolean flag indicating whether the sender is a *RGCH*.

***DATA_REQUEST* message**

This message is sent by the MS when it reaches a VP to request collected data from RGCHs. The message is forwarded hierarchically from the MS to the adjacent RGCHs, then to the associated GCHs, and finally to the respective SNs. It includes the following fields:

- *IdSender:* Identifier of the sender (MS, RGCH or GCH).
- *IdReceiver:* Identifier of the receiver (RGCH, GCH or SN).
- *ListRGCHs:* List of RGCHs requested by the MS at a given VP.

***DATA* message**

SNs communicate their gathered data to their GCHs using this message, or RGCHs use it in response to `DATA_REQUEST` requests from the MS. It includes the following fields:

- *IdSender:* Identifier of the sender (SN, GCH or RGCH).
- *IdReceiver:* Identifier of the receiver (GCH, RGCH, or MS).
- *CollectedData:* The actual sensor readings being transmitted.

***GCH_REELECTION* message**

A GCH broadcasts this message when its remaining energy hits a certain level, which triggers the **re-election process**. It contains the following fields:

- *GCH ID:* Identifier of the GCH initiating re-election.
- *Cell ID:* Identifier of the GCH's cell.
- *RE:* Remaining energy level of the GCH.

Algorithm 1: Grid Construction Process

Input: L, R_{max}

Output: A constructed $K \times K$ virtual grid with node-cell assignments

1 Compute the grid adjustment factor using formula 4.3;

2 Determine the value of k based on the value of a using formula 4.4;

3 Compute the cell side length a using formula 4.5;

`// Assigning Sensor Nodes to Grid Cells`

4 **foreach** *sensor node i with coordinates* (x_i, y_i) **do**

5 Node i calculates the identifier of its corresponding cell using formula 4.6;

6 **end**

4.4.3 Virtual grid construction

The virtual grid design approach organizes the sensor network into an organized grid, improving the efficiency of data collection and communication. It ensures single-hop transmission between adjacent cells by partitioning the sensing region into square cells based on the communication range of SNs. Algorithm 1 illustrates the grid construction process.

The following steps must be followed in order to create a virtual grid with dimensions of $k \times k$:

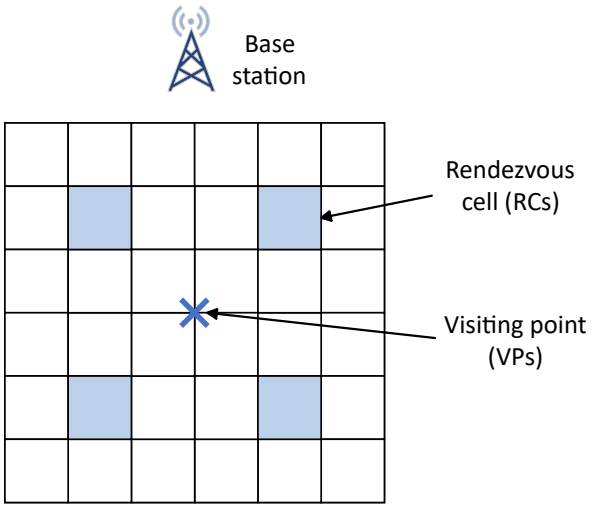
1. **The determination of the value of k:**

   - k must be a multiple of 3 (e.g., 6, 9, 12, 15, etc.) to simplify the selection of RCs and VPs, as illustrated in the Figure (4.1).

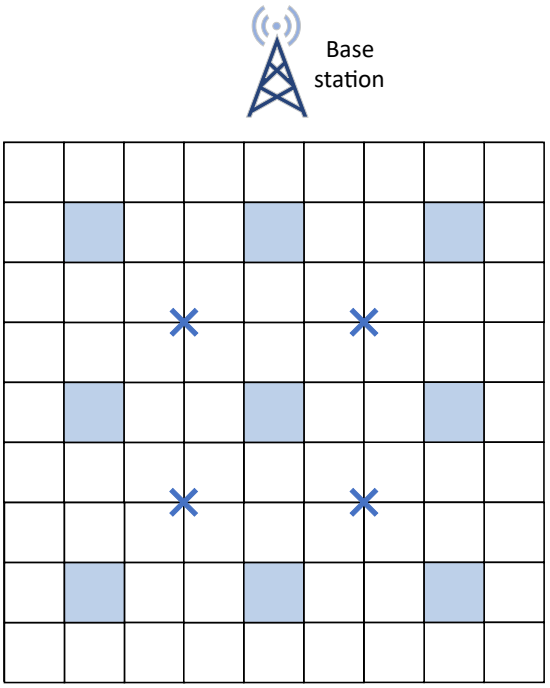
   - Two nodes located in adjacent cells must be able to communicate, ensuring that they fall within each other's communication range.

   - k is determined based on the terrain side length L and the communication range of SNs R_{max} using the adjustment factor α which is calculated as follows:

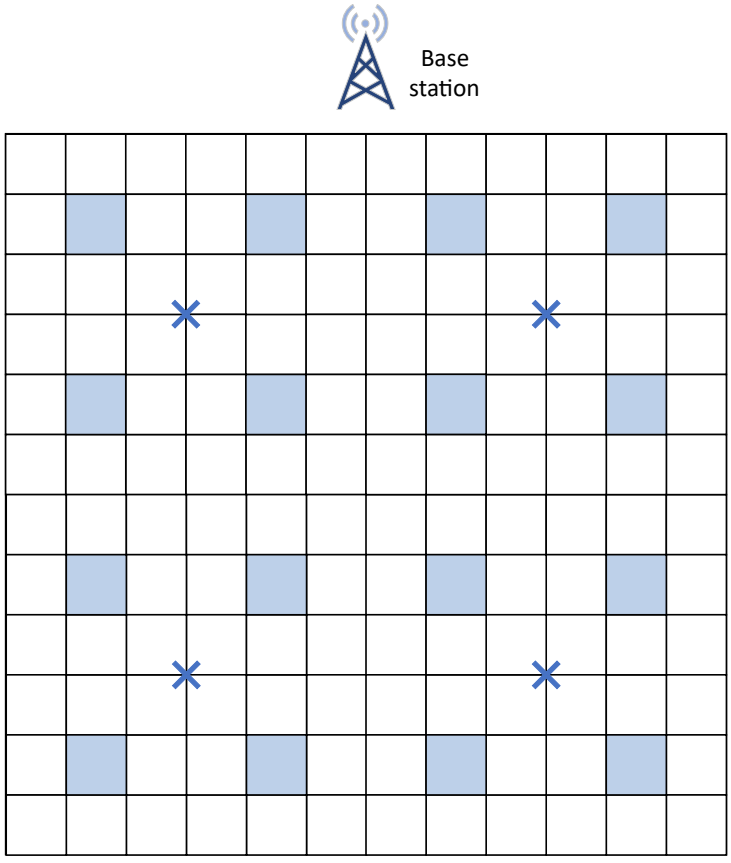
$$\alpha = \frac{2\sqrt{2}L}{R_{max}} \tag{4.3}$$



(a) 6 × 6 virtual grid.



(b) 9 × 9 virtual grid.



(c) 12 × 12 virtual grid.

Figure 4.1: Diverse constructed virtual grids along with their respective RCs and VPs.

Based on the value of α, k is chosen as follows:

$$k = \begin{cases} 6, & \text{if } \alpha \leq 6 \\ 9, & \text{if } 6 < \alpha \leq 9 \\ 12, & \text{if } 9 < \alpha \leq 12 \\ 15, & \text{if } 12 < \alpha \leq 15 \\ \ldots & \ldots \end{cases} \tag{4.4}$$

2. **Cell side length:** The side length of each cell, a, is computed as follows:

$$a = \frac{L}{k} \tag{4.5}$$

Figure (4.2) illustrates the rationale underlying the construction of the virtual grid.

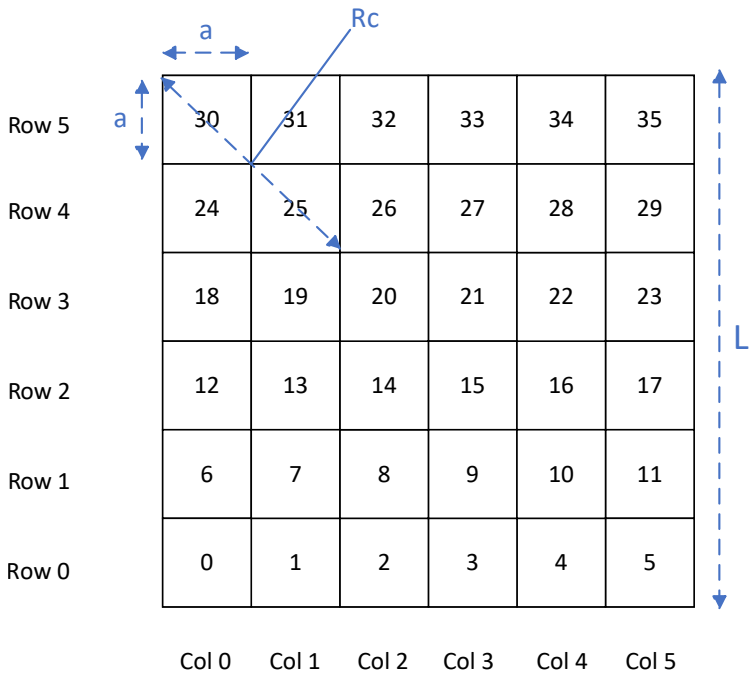


Figure 4.2: Grid construction principle.

3. **Cell identifier calculation:** Each SN i computes its cell identifier $IdCell_i$ based on its coordinates (x_i, y_i) as follows:

$$IdCell_i = p \times k + q \tag{4.6}$$

where:

- $p = \lfloor \frac{y_i}{a} \rfloor \rightarrow$ column number of the cell.
- $q = \lfloor \frac{x_i}{a} \rfloor \rightarrow$ row number of the cell.

4.4.4 Determination of RCs

To optimize data collection and reduce communication overhead, a set of RCs is selected from the constructed virtual grid. The goal is to allow the RGCHs to gather data from the GCHs of the eight surrounding regular cells and transmit an aggregated message to the MS. The identification of these RCs is governed by the following formula:

$$m = 3z + 1 \quad \text{where } z = 0, 1, 2, \ldots, \left(\frac{k}{3} - 1\right) \tag{4.7}$$

The different values of m denote the potential row or column coordinates of the selected RCs, calculated based on various values of z. The algorithm 2 describes this RCs identification process.

Algorithm 2: Selection of RCs

Input: k

Output: $rCells$: List of RCs' identifiers

```
1  rCells ← ∅;
2  for i ← 0 to (k/3 − 1) do
3      x ← 3 × i + 1;
4      for j ← 0 to (k/3 − 1) do
5          y ← 3 × j + 1;
6          idOfCHCell ← y × k + x;
7          rCells ← rCells ∪ {idOfCHCell};
8      end
9  end
10 return rCells;
```

Upon identification of these cells, the GCHs of these cells become RGCHs. Subsequently, they communicate their new role to the neighboring GCHs through a specific control message.

Figure 4.3 illustrates the identified RCs within a 9×9 grid based on this formula. In this figure, i and j can take the same values as z.

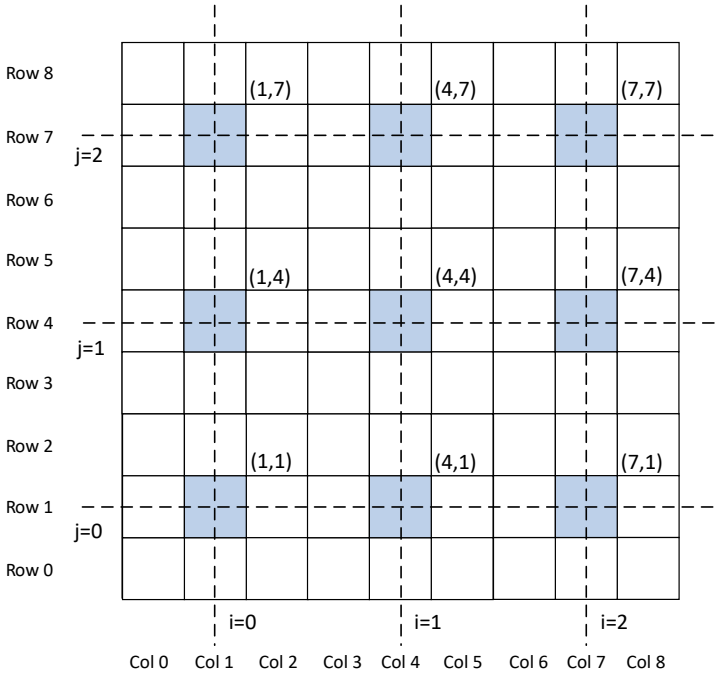


Figure 4.3: RCs identification in a 9×9 grid.

4.4.5 Election of GCHs and RGCHs

The next step after constructing the virtual grid is to elect a GCH for each cell and establish clusters. The following steps comprise the election process:

- **Determine the exact center of the cell:** If the cell has coordinates ranging from $(x_{\min}, y_{\min})$ to $(x_{\max}, y_{\max})$, the center c is computed as:

$$X_c = \frac{x_{\min} + x_{\max}}{2}, \quad Y_c = \frac{y_{\min} + y_{\max}}{2} \tag{4.8}$$

- **Find the SN closest to (X_c, Y_c):** The Euclidean distance $\delta_{i,c}$ from each node i in the cell to the center is then calculated. The node with the smallest distance $\delta_{i,c}$ is selected as the GCH, as illustrated in Figure (4.4).

- **Ensure the selected node has sufficient energy:** The selected node must have a residual energy level above a predefined threshold $E_{threshold}$.

- **Announce the GCH role:** The selected GCH broadcasts a *GCH_ANNOUNCEMENT* message to notify all nodes within its cell of its new role.

This election process is demonstrated in Algorithm 3.

Algorithm 3: GCHs election process within cells

Input: S_{cell_k}: Set of sensor nodes in the cell k

$E_{\text{threshold}}$: Minimum required residual energy to ensure the GCH role

Output: GCH: The elected Cluster Head that is closest to the cell center c

1 The coordinates of c of the cell k are calculated using formula 4.8;

// Find the nearest node to c

2 $\delta_{\min} \leftarrow \infty$;

3 **foreach** *node* $i \in S_{cell_k}$ **do**

4 Compute $\delta_{i,c}$ using formula 4.2;

5 **if** $\delta_{i,c} < \delta_{\min}$ **then**

6 $GCH \leftarrow s$;

7 $\delta_{\min} \leftarrow \delta_{i,c}$;

8 **end**

9 **end**

// Ensure the Selected Node has Sufficient Energy

// $E_{\text{Res}}(GCH)$ returns the residual energy of the GCH

10 **if** $E_{Res}(GCH) < E_{threshold}$ **then**

11 Remove GCH from candidates nodes;

12 **goto** 2;

13 **end**

// Announce the GCH Election

14 Broadcast *GCH_ANNOUNCEMENT* message;

4.4.6 Determination of VPs

Following the identification of the RCs, the VPs must be chosen in order to optimize the MS's mobility for data gathering. Within the sensing region, a VP denotes a spot where the MS stops to collect information from many RGCHs. The VPs of the MS are chosen from the intersection points of four contiguous cells. The selection is designed so that each VP enables

the MS to request data from four nearby RGCHs in a single hop, with care to minimize the necessity to request data from the same RGCHs over distinct VPs where feasible. The formula used to determine these points is as follows:

$$m = a(3z+3) \quad \text{where} \quad z = 0,2,4,\ldots,\left(\frac{k}{3}-2\right) \tag{4.9}$$

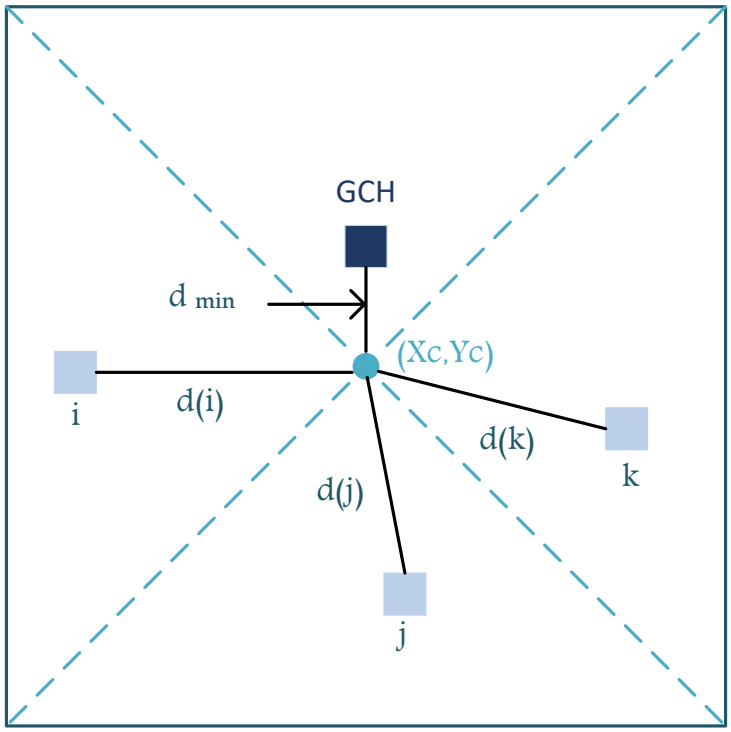
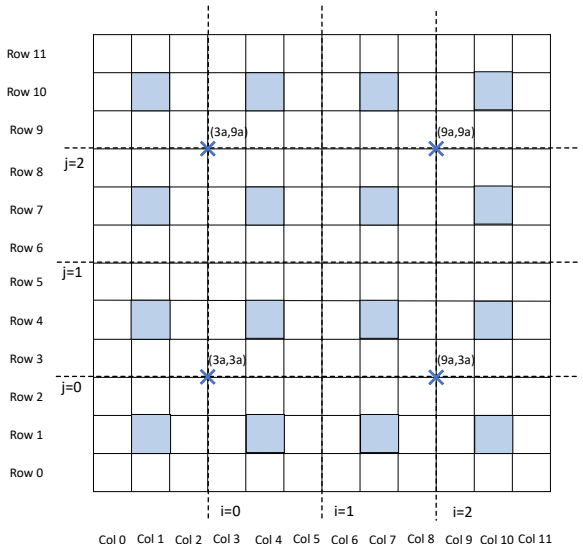


Figure 4.4: Selection of the closest SN to the center c having the coordinates (X_c, Y_c).

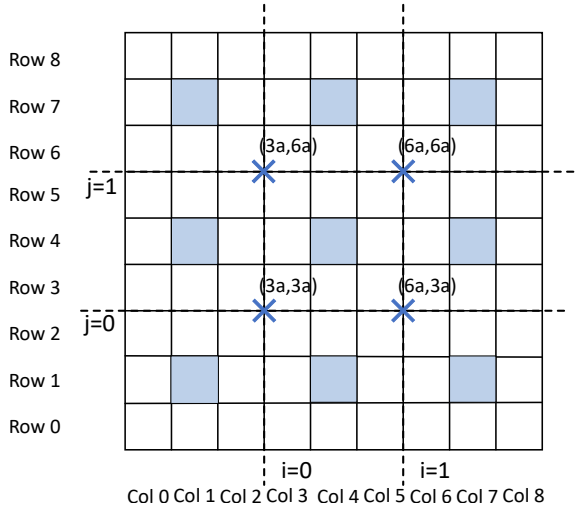
Figures (4.5b), (4.5a), and (4.5c) present how these VPs are identified within 9×9, 12×12, and 15×15 grids, respectively. Just to further simplify, if k = 9, z = 0 or 1; if k = 12, z = 0 or 2; and if k = 15, z = 0, 2, or 3. In what follows, the set of selected VPs is denoted as θ.

By carefully selecting these VPs, the MS may effectively gather data from four nearby RGCHs in a single hop, reducing redundant data transmissions and minimizing energy consumption.

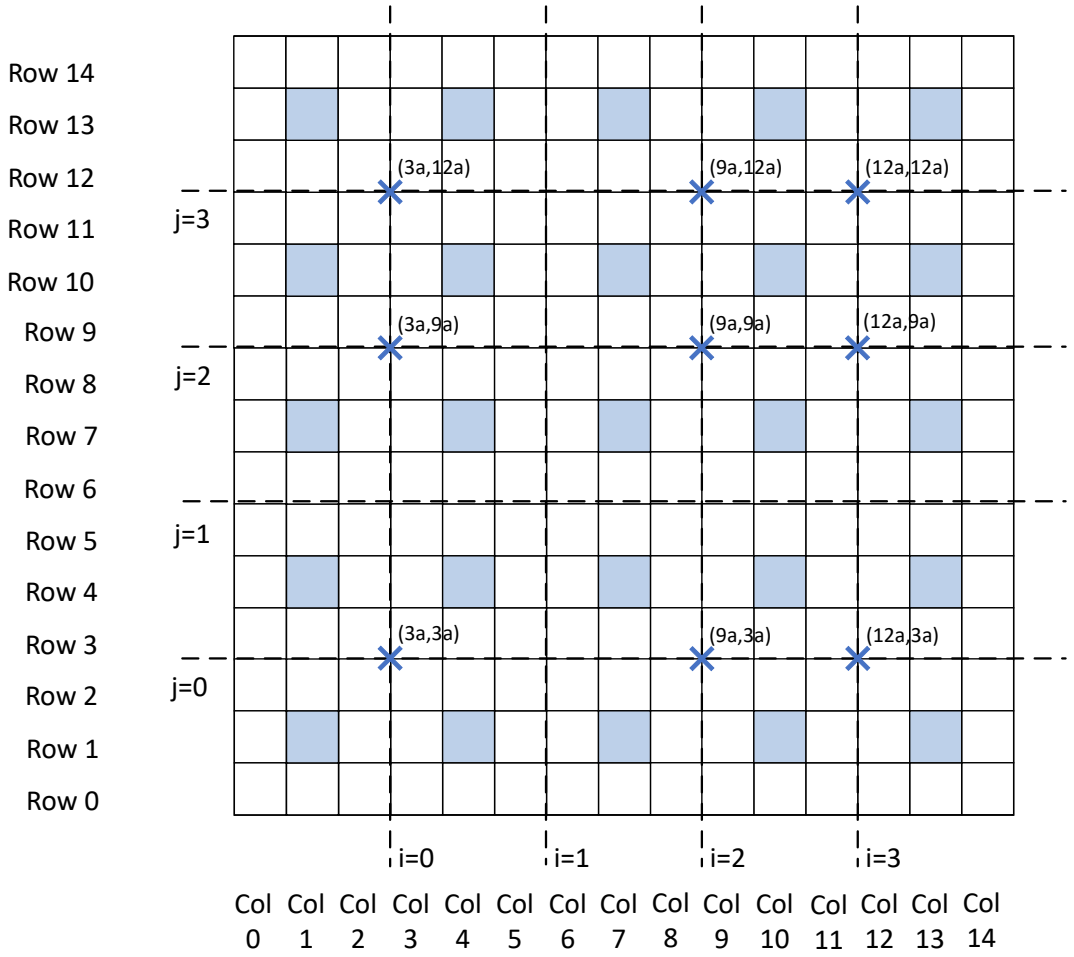
This systematic selection of VPs ensures that the MS follows an optimized trajectory, limiting unnecessary movements while maintaining network performance and extending the network lifetime. Algorithm 4 provides a detailed illustration of this selection process.



(a) VPs identification for even k ($k = 12$).



(b) VPs identification for $k = 9$.



(c) VPs identification for $k = 15$.

Figure 4.5: Comparison of VPs identification for various grid sizes.

Algorithm 4: Selection of VPs

Input : L, a, k

Output: θ: the list of coordinates of the identified VPs initialized to $\emptyset$

```
// The BS coordinates, corresponding to the point where the MS
   starts and ends its journey, are added
```

$\theta \leftarrow \theta \cup \{(\frac{L}{2},0)\}$;
$i \leftarrow 0$;
while $i \leq \lfloor \frac{k}{3} \rfloor - 2$ **do**
    $x \leftarrow a \times (3 \times i + 3)$;
    $j \leftarrow 0$;
    while $j \leq \lfloor \frac{k}{3} \rfloor - 2$ **do**
        $y \leftarrow a \times (3 \times j + 3)$;
        $\theta \leftarrow \theta \cup \{(x,y)\}$;
        if $j = \lfloor \frac{k}{3} \rfloor - 3$ **then**
            $j \leftarrow j + 1$;
        else
            $j \leftarrow j + 2$;
        end
    end
    if $i = \lfloor \frac{k}{3} \rfloor - 3$ **then**
        $i \leftarrow i + 1$;
    else
        $i \leftarrow i + 2$;
    end
end
return θ;

4.5 MS's path planning

As indicated earlier, the MS visits all of the selected VPs along a predefined path, beginning and ending its journey at the BS. In order to guarantee that every VP is visited precisely once during the MS's traverse, the BS chooses one of the potential Hamiltonian cycles.

To construct this path, the BS models the MS's movement using a connected graph $G = \{V, E\}$, as shown in Figure (4.6) where:

- $V = \theta \cup \{\text{BS}\}$ represents the set of vertices, including all VPs and the BS.
- E is the set of edges connecting these vertices.

Because the Hamiltonian cycle issue is NP-complete, the Backtracking algorithm is used as a practical solution approach. This algorithm effectively finds a feasible route for the MS often producing good results.

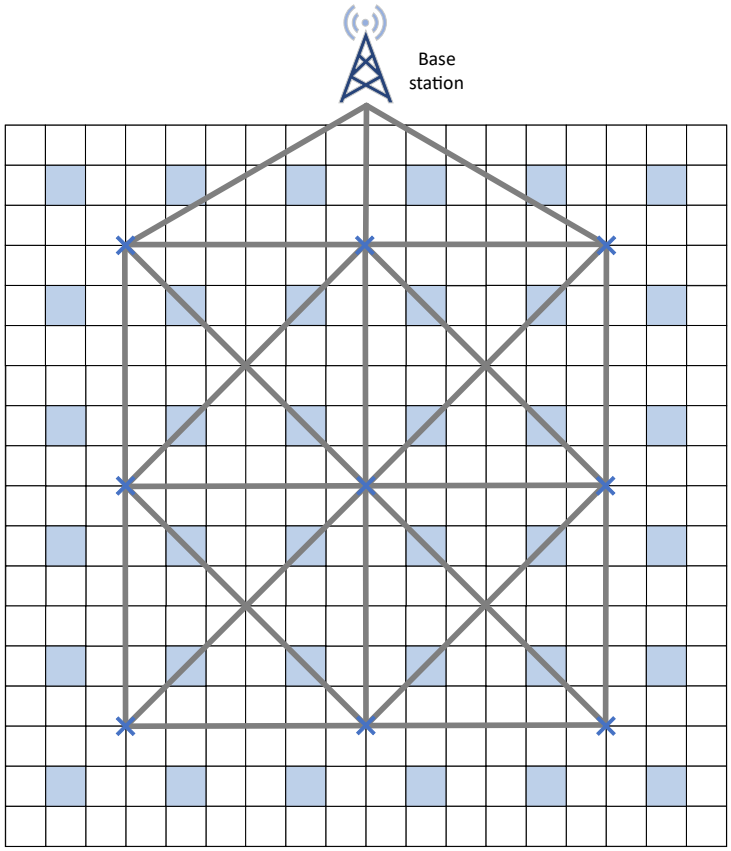


Figure 4.6: Connected graph of VPs with a 18×18 grid.

4.6 Data collection process

The sink node regularly gathers data, adhering to one of the Hamiltonian cycles established by the BS. Upon the MS's arrival at a VP, it broadcasts a *DATA_REQUEST* message using its maximum transmission power. This request is forwarded hierarchically from the MS to the adjacent RGCHs, then to the associated GCHs, and finally to the corresponding SNs. Each SN,

upon receiving the request, prepares a *DATA* message containing its sensed data. This message is then sent in the reverse direction: from the SN to its GCH, which aggregates it and forwards it to the respective RGCH, and finally to the MS. The MS continues its journey by employing the same process until it returns to the BS. Figure 4.7 illustrates this collection process.

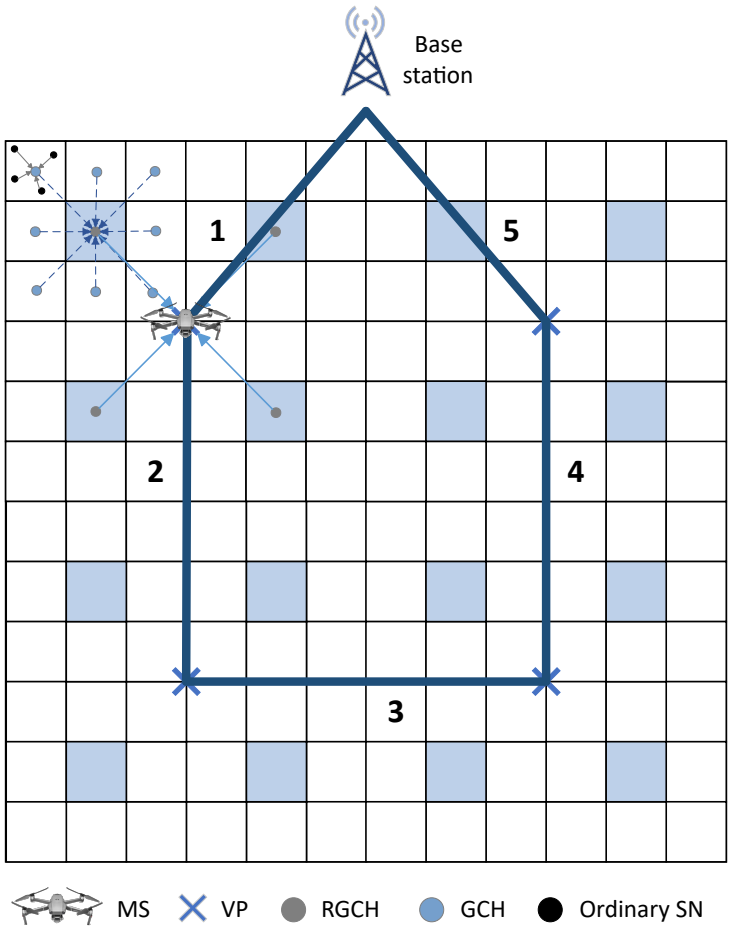


Figure 4.7: Path planning and data collection.

4.7 Re-election of GCHs

When the residual energy of a GCH falls below $E_{threshold}$, it initiates a re-election process by broadcasting a *GCH_RE-ELECTION* message. The current GCH is excluded from the re-election process, and a new GCH is selected following the same election process as depicted in section 4.4.5.

4.8 Conclusion

This chapter provided a comprehensive overview of the proposed protocol EDCVP including its design and functionality. It started with a summary of the chosen network model and key assumptions, followed by a description of the approach, which included key data structures, message formats, and the main procedural steps. This chapter prepares for the validation phase, with the following chapter focusing on the presentation and analysis of simulation results to assess the EDCVP protocol's effectiveness and performance.

Chapter 5

Performance Evaluation

5.1 Introduction

This chapter aims to validate our proposed EDCVP protocol through simulations using the NS-3 simulator. The benchmark protocol EDEDA was also implemented and evaluated under the same conditions to ensure a fair performance comparison. The chapter begins by reviewing the NS-3 environment and explaining the reasons for selecting it for this study. The chapter then outlines the steps taken to create simulated scenarios, apply the two protocols, and assess performance using specified criteria. This chapter discusses important elements, including the employed energy model, the selected simulation parameters, the comparison metrics utilized, and the obtained results. The objective is to determine whether EDCVP improves energy efficiency, reduces data collection delay, and enhances overall network lifetime compared to EDEDA.

5.2 The ns-3 simulator

NS-3 is a communication network simulator built for teaching and research using discrete-event models [3]. It is freely available under the GNU General Public License (GPLv2), permitting open-source development and community collaboration. It is a recent, versatile tool for modeling complicated network behaviors. It is modular in design and supports both Python and C++.

NS-3 was used for this study due to its ability to accurately simulate the behavior of computer networks, including WSNs. It offers support for mobility models, energy-efficient components, and configurable routing protocols attributes crucial for the precise assessment of our MS-based collection approach. Furthermore, NS-3 facilitates the validity and reproducibility of simulation results, which corresponds with the scientific objectives of this study.

To implement and evaluate our proposed protocol alongside EDEDA, the comparative protocol, we followed a structured process within the NS-3 environment:

- **Protocol implementation:** the specifications of both protocols including, the different packets used, the sensor nodes and MS behaviors, were implemented using C++, following NS-3's architecture and design standards. The reader may refer to Appendix A for a clearer understanding of the implementation of our solution.
- **Scenario configuration:** we defined the simulation parameters in configuration scripts, encompassing the number of deployed nodes, dimensions of the deployment region, transmission range, sink mobility pattern, and other relevant factors. The section A.1 illustrate our simulation setup file.
- **Visualization:** using NetAnim, NS-3's official visualization tool, we monitored the simulation in real time, primarily observing node placement and the mobility trajectory of the MS. It is worth noting that NetAnim can also display key performance indicators such as throughput, latency, and packet loss.
- **Result analysis:** after the simulation, output files generated by NS-3 were analyzed to assess the protocol's performance. Metrics like energy consumption, data collection delay, and network lifetime were used to support our evaluation and conclusions.

5.3 Simulation methodology

We implemented our proposed EDCVP protocol into the NS-3 simulator to evaluate its performance. As mentioned before, we also implemented the EDEDA protocol [4], which shares similarities with our case by employing a virtual grid and MS for data collection. To guarantee a fair comparison and assess the enhancements introduced by EDCVP, both protocols were simulated under identical conditions.

5.3.1 Simulation parameters

Several simulation tests were performed to evaluate the performance of the EDCVP protocol under different network settings once it was included in the NS-3 simulator. From 100 to 400 SNs, we conducted the simulations to see how the density of the network affected the lifetime and energy consumption of the network. To further assess how different terrain dimensions (value of L) affected data collection time, we adjusted the deployment area size (the value of L). The consistency of the results was ensured by keeping many critical parameters constant across all simulation runs. Table 5.1 provides a summary of the simulation parameters that were used throughout our evaluation.

Table 5.1: Simulation parameters.

Parameter	**Value**
Terrain dimensions ($L \times L$)	240×240, 280×280, 320×320, 360×360,400×400
Number of SNs	200-400
Communication range (R_{max})	80 m
Initial residual energy	1 J
Energy threshold (ratio)	0.4
Packet size	512 Bytes
Mobile sink speed	10 m/s
Number of sink tours	90
Sink sojourn time at a VP	10 s
Duration between two tours	60 min
Simulation duration	96 hours
E_{elec}	50 nJ/bit
E_{mp}	0.0013 pJ/bit
E_{fs}	10 pJ/bit

To simulate a real-life scenario, we examined a WSN established to detect likely fire incidents utilizing an unmanned aerial vehicle, such as a drone. SNs are evenly distributed over the sensing area. The drone initiates its route from the base station, traversing to specified VPs to gather sensed data from the associated SNs, before returning to the base station for subsequent analysis of the data. The drone conducts a data collection expedition every hour. A total of 90 trips were evaluated, amounting to a monitoring period of about four days.

5.3.2 Energy model

Our network's energy consumption was evaluated using the radio energy dissipation model as described in [41]. From a given distance d, this model determines both the energy needed to send and receive a message of p bits. The transmission energy, $E_{TX}(p,d)$, and the reception energy $E_{RX}(p)$ are defined in the following way:

$$E_{TX}(p,d) = \begin{cases} p \cdot E_{elec} + p \cdot E_{fs} \cdot d^2, & \text{if } d < d_0 \\ p \cdot E_{elec} + p \cdot E_{mp} \cdot d^4, & \text{if } d \geq d_0 \end{cases} \tag{5.1}$$

$$E_{RX}(p) = p \cdot E_{elec} \tag{5.2}$$

The components of the above energy consumption formula are described as follows:

- $E_{TX}(p,d)$ Represents the total energy required to transmit a message of p bits over a distance d.
- $E_{RX}(p)$ corresponds to the total energy consumption when receiving a message of p bits.
- E_{elec} is the energy consumed by the sensor's internal electronics during the transmission process.
- E_{fs} is the energy required to transmit a single bit under free-space propagation conditions.
- E_{mp} is the energy required to transmit a single bit under multipath propagation conditions.
- d_0 is the threshold distance that separates the two propagation models: free space (for short distances) and multipath (for longer distances).

5.3.3 Benchmark protocol

We use EDEDA [4] as the reference protocol to evaluate the performance of the suggested EDCVP protocol. Its principle was previously outlined in section 3.10. EDEDA was selected because of its architectural similarity to our method, especially its use of a virtual grid-based structure and an MS for data collection. Like EDCVP, EDEDA depends on the idea of VPs where the sink stops to gather data from adjacent SNs. Regarding energy efficiency, collection latency, and general network performance, this common design concept makes a fair and significant comparison possible. Moreover, EDEDA has shown excellent results in past research, particularly in terms of minimizing latency and balancing energy use, which fit the aims of our suggested approach.

5.3.4 Comparison metrics

To evaluate the performance of the proposed EDCVP protocol, we selected a set of metrics that allow for direct comparison with the reference protocol, EDEDA. These metrics reflect key performance aspects in WSNs, including total energy consumption and network lifetime. Furthermore, when employing a MS with predefined trajectories, metrics such as data collection latency, number of VPs, and tour length become especially pertinent. The details related to each metric are presented in the following sections.

5.3.4.1 Total energy consumption

This indicator shows how much energy the network uses overall during the communication phase, which covers both data transmission and reception. If the starting energy level of each SN i is E_{init_i} and its remaining energy at the end of the simulation is E_{resid_i} (determined using the previously mentioned energy model), then the total energy spent by all n nodes in the network can be estimated as follows:

$$E_{total} = \sum_{i=1}^{n} [E_{init_i} - E_{resid_i}] \tag{5.3}$$

5.3.4.2 Network lifetime

Defined as the network's deterioration to a level where it can no longer fulfil its intended purpose. This may occur under any of the following circumstances: the death of the first sensor node, the death of a specified number or percentage of nodes, the partitioning of the

network resulting in a lack of connection across subnetworks, or the loss of coverage [49]. For this evaluation phase, we consider the initial event that is often used. In particular, the network lifetime refers to the time elapsed from the start of the simulation until the first node completely depletes its energy. This gives an indication of how well the protocol balances energy consumption across the network.

5.3.4.3 Data collection delay

It is defined as the total time required for the MS to begin its journey from the BS, visit all VPs to collect data, and return to the BS to deliver the collected data. This delay includes both the travel time between VPs and the data transmission time at each point, which corresponds to the pause time. The data collection time is given by the following formula:

$$D_{\text{collection}} = \sum_{i=1}^{|VP \cup \{\text{Start Point}\}|} \frac{\delta(VPL(i), VPL(i+1))}{v} + \sum_{j=1}^{|VP|} Pt_j \tag{5.4}$$

Where

- VP: Set of visiting points.
- $VPL(i)$: Location of the i^{th} visiting point.
- $\delta(VPL(i), VPL(i+1))$: Euclidean distance between two consecutive VPs.
- v: Speed of the MS.
- Pt_j: Pause time at the j^{th} visiting point.
- Start Point: The base station or starting position of the MS.

5.3.4.4 Number of VPs

This metric indicates the total number of VPs selected by the protocol. A lower number of VPs can reduce overall movement, hence reducing data collection latency.

5.3.4.5 MS tour length

This metric represents the MS's overall distance covered on one full data-collecting tour. It comprises the total Euclidean distances between all the following VPs along the planned

Hamiltonian cycle, starting and finishing at the BS. The MS tour length is a crucial metric for evaluating mainly the time efficiency of sink mobility planning. Reducing the duration of a trip helps minimize delays while collecting data.

5.4 Results and discussion

To demonstrate the efficiency of our proposed protocol, we present the obtained results according to the selected performance metrics as follows:

- First, we evaluate the impact of network density on energy consumption. A graph is used to show the total energy consumed as the number of SNs increases for both EDCVP and EDEDA protocols.
- Next, we analyze how network density influences the network lifetime. This is illustrated through a graph that plots the network lifetime against the number of SNs for both protocols.
- Finally, we study the effect of terrain size (value of L) on the total tour length and data collection delay of the MS. The results are shown using graphs that relate both metrics to different terrain sizes.

To guarantee more reliable findings, each point on the graph indicates the average of 10 experiments performed with varying random seed values. This method enabled us to examine the performance of both protocols across different network topologies.

5.4.1 Energy consumption

In this first analysis, we evaluate the impact of network density on energy consumption using the EDCVP and EDEDA protocols. A series of simulations were conducted under identical conditions using varying numbers of SN. The total energy consumed by the network was recorded for each scenario, ranging from 200 to 400 nodes, while keeping the terrain size fixed at 240×240 meters. The energy consumption results, as illustrated in Figure 5.1, demonstrate that the total energy usage increases progressively with the number of SN in both EDCVP and EDEDA protocols. This behavior is predicted as more nodes produce more data, thereby increasing packet transmissions and communication overhead. However, the total en-

ergy consumed in the case of EDCVP is slightly lower than that consumed by EDEDA. For example, with 200 SN, EDCVP consumes approximately 136.85 joules, whereas EDEDA consumes around 145.06 joules, a reduction of about 5.9%. This efficiency persists at higher densities; with 400 nodes, EDCVP uses 293.09 joules, while EDEDA reaches 302.06 joules, reflecting a 3% improvement. This performance gain is primarily due to the reduced use of control messages, particularly data request messages. In both protocols, the MS broadcasts an DATA_REQUEST message each time it halts at a given VP. However, EDCVP strategically selects fewer and better-placed VPs, one per 36 cells compared to one per 9 in EDEDA, resulting in fewer MS halts, fewer control messages, and broader data coverage per VP. Additionally, EDCVP implements multi-level data aggregation, as shown in Figure 5.2a where data is first aggregated at the GCH level (red nodes) and then again at the RGCH level (green nodes) before being sent to the MS. This layered aggregation significantly reduces communication overhead, contributing further to the energy savings while minimizing the number of VPs.

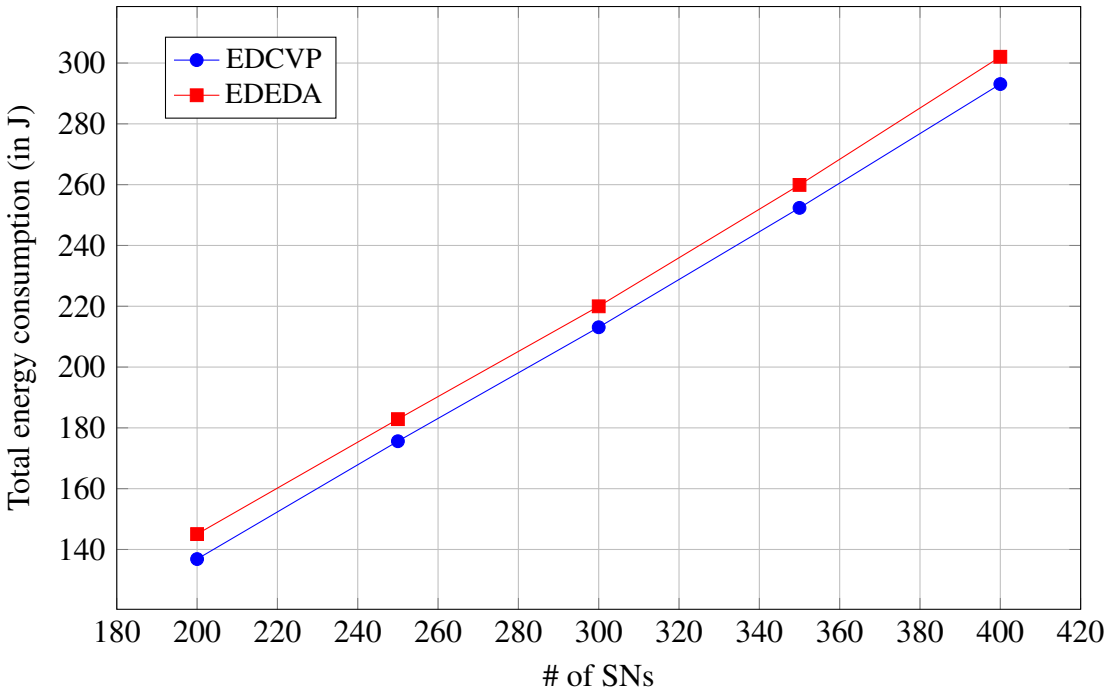
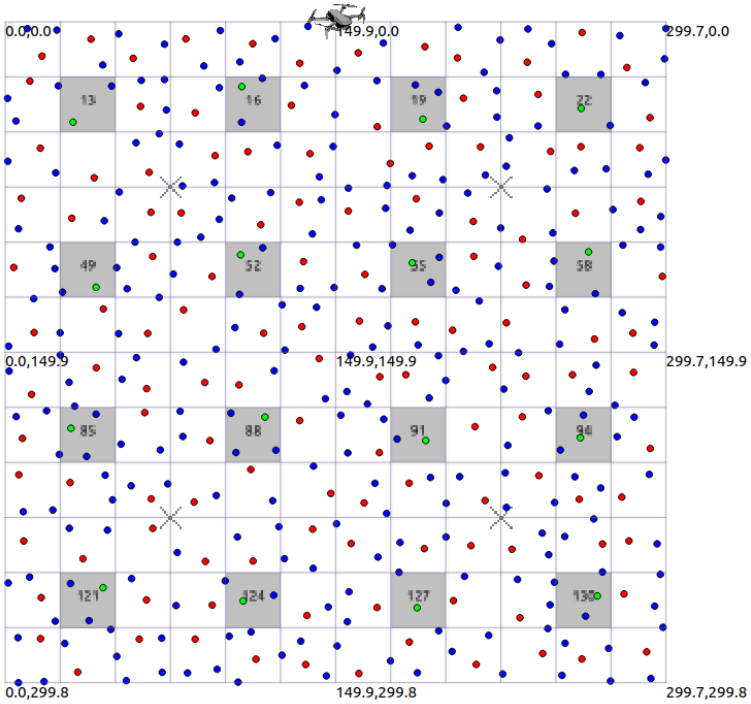
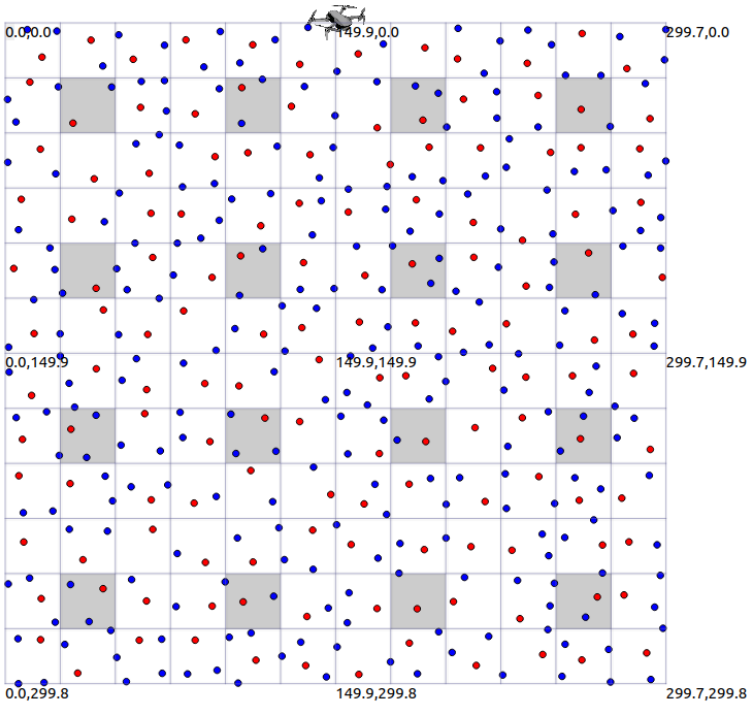


Figure 5.1: Comparison of EDCVP and EDEDA: Number of SNs vs Total energy consumption.



(a) EDCVP



(b) EDEDA

Figure 5.2: Visual comparison of data aggregation levels and VPs selection in EDCVP and EDEDA, as captured from NetAnim.

5.4.2 Network lifetime

In this second analysis, we evaluate the impact of network density on network lifetime using the EDCVP and EDEDA protocols. This evaluation was carried out in the same simulation environment used for the energy consumption analysis, with the number of SN varying from 200 to 400 while keeping the terrain size fixed at 240×240 meters. As mentioned before, the network lifetime was measured as the time elapsed from the start of the simulation until the first node completely depleted its energy. This metric provides an indication of how well the protocol balances energy consumption across the network. The results, illustrated in Figure 5.3, depict the network lifetime as a function of the number of SN for both protocols. The results show a consistent decrease in network lifetime as the number of SNs increases. This behavior is expected and can be attributed to the increasing node density: maintaining a constant value of L (240 meters) while increasing the number of nodes results in an elevated node density. This higher density leads to increased overall communication activity and energy consumption across the network. Nodes, particularly GCHs and RGCHs in EDCVP and GCHs in EDEDA, are more heavily utilized, as they are responsible for aggregating and forwarding significant volumes of data. This elevated communication load accelerates battery depletion in these key nodes, resulting in earlier node failures and a shortened overall network lifetime. Despite this general decline, EDCVP consistently outperforms EDEDA in terms of network lifetime. For instance, with 200 SNs, EDCVP maintains a lifetime of approximately 86.29 hours, while EDEDA reaches only 78.50 hours, an improvement of about 9.9%. This advantage persists with increasing density; at 400 nodes, EDCVP achieves 76.16 hours compared to 70.10 hours in EDEDA, representing an 8.6% increase. Since the total energy consumption is higher in EDEDA, the communication load on nodes is greater. This increased load accelerates battery depletion, leading to the premature death of SNs, consequently, a shorter network lifetime. Both protocols employ a dynamic GCH re-election mechanism, allowing periodic rotation of cluster head roles to distribute energy consumption more evenly across nodes. This helps delay the failure of individual nodes by preventing prolonged overuse.

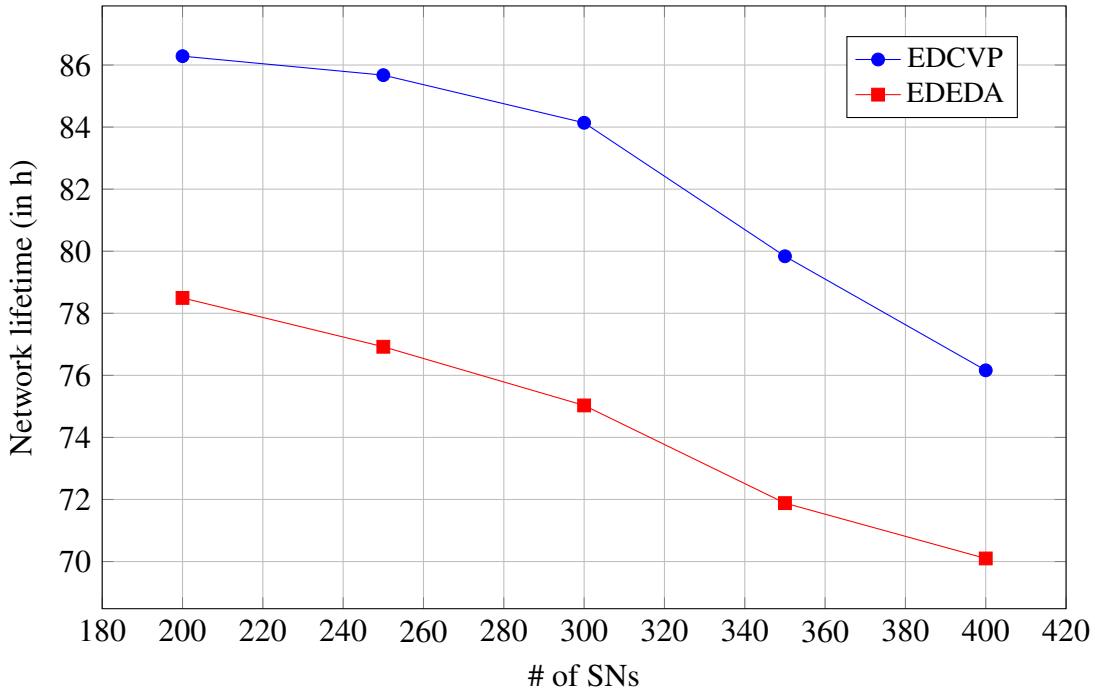


Figure 5.3: Comparison of EDCVP and EDEDA: Number of SNs vs Network lifetime.

5.4.3 MS tour length & Data collection delay

In this final analysis, we focus on two closely related performance metrics: the MS's tour length and the corresponding data collection delay. These metrics were evaluated in the same simulation environment, where the terrain size was varied from 240×240 to 440×440 meters, with a fixed number of 400 SNs deployed uniformly across the area. The MS followed a predefined Hamiltonian cycle that passed through all selected VPs, collecting data during each stop. The sojourn time at each VP was set to 10 seconds, and the MS maintained a constant speed throughout its journey (10 m/s). The results presented here provide a clear comparison of how the two protocols, EDCVP and EDEDA, perform in terms of minimizing MS movement and reducing total data collection time. The MS's tour length and the resulting data collection delay are inherently linked. Longer MS paths result in proportionally longer data collection delays. In our proposed EDCVP protocol, both metrics are significantly improved compared to EDEDA. The primary reason for this improvement is the smaller number of VPs used in EDCVP. As presented in Table 5.2, with a grid size of 9×9, EDCVP uses only 4 VPs, while EDEDA uses 9. With a 12×12 grid, as presented also in Figure 5.2, EDCVP again uses 4 VPs, whereas EDEDA uses 16. And for a 15×15 grid, EDCVP uses 9 VPs compared to 25 in EDEDA. These differences directly reduce the number of stops the MS must make, which

shortens the tour and lowers the data collection delay. This effect is visualized in Figure 5.4 and Figure 5.5:

- For example, at a terrain size of 240×240, EDCVP selects 4 VPs, and the MS completes its tour in 418.885 meters with a delay of 1.36 minutes. In contrast, EDEDA selects 9 VPs, and the MS travels 885.16 meters with a delay of 2.97 minutes. This represents a 111.26% increase in tour length and a 118.01% increase in delay for EDEDA compared to EDCVP.

- At a terrain size of 400×400, EDCVP selects 9 VPs, and the MS travels 1490.03 meters in 3.98 minutes. In comparison, EDEDA selects 25 VPs, and the MS covers 2448.67 meters in 8.24 minutes. This corresponds to a 64.36% longer path and a 107.02% increase in delay for EDEDA.

These consistent improvements across all terrain sizes highlight the benefits of EDCVP's optimized VP selection. Fewer and more strategically placed VPs result in reduced MS travel and

more efficient overall data collection.

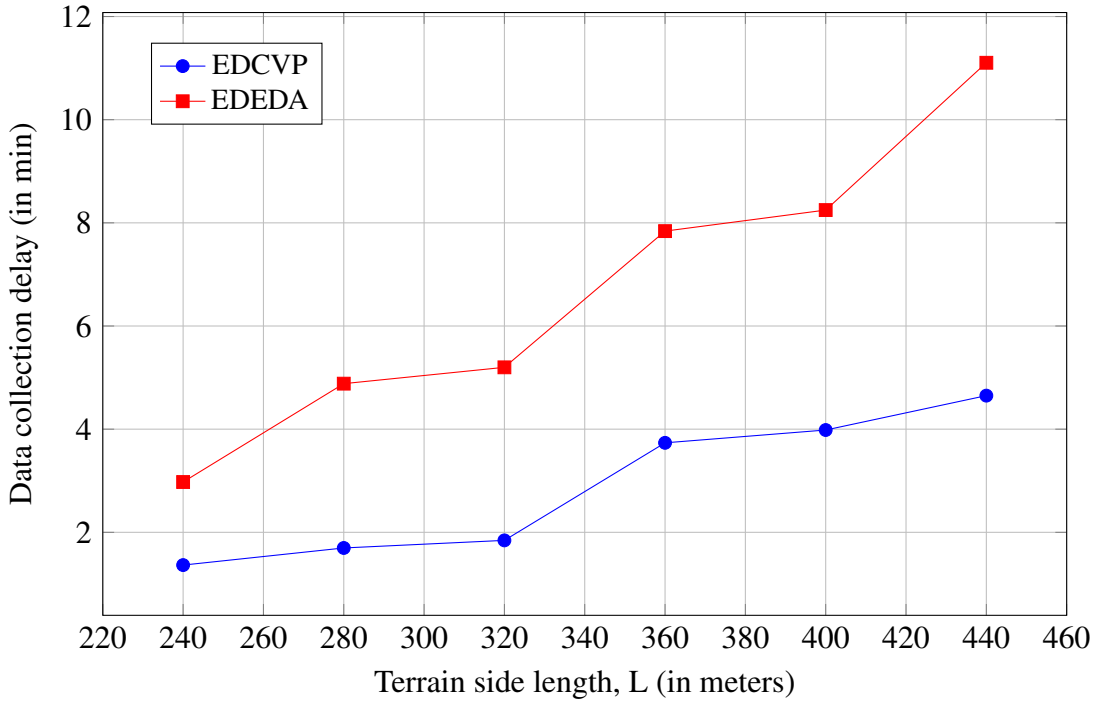


Figure 5.4: Comparison of EDCVP and EDEDA: Terrain side length (L) vs Data collection delay.

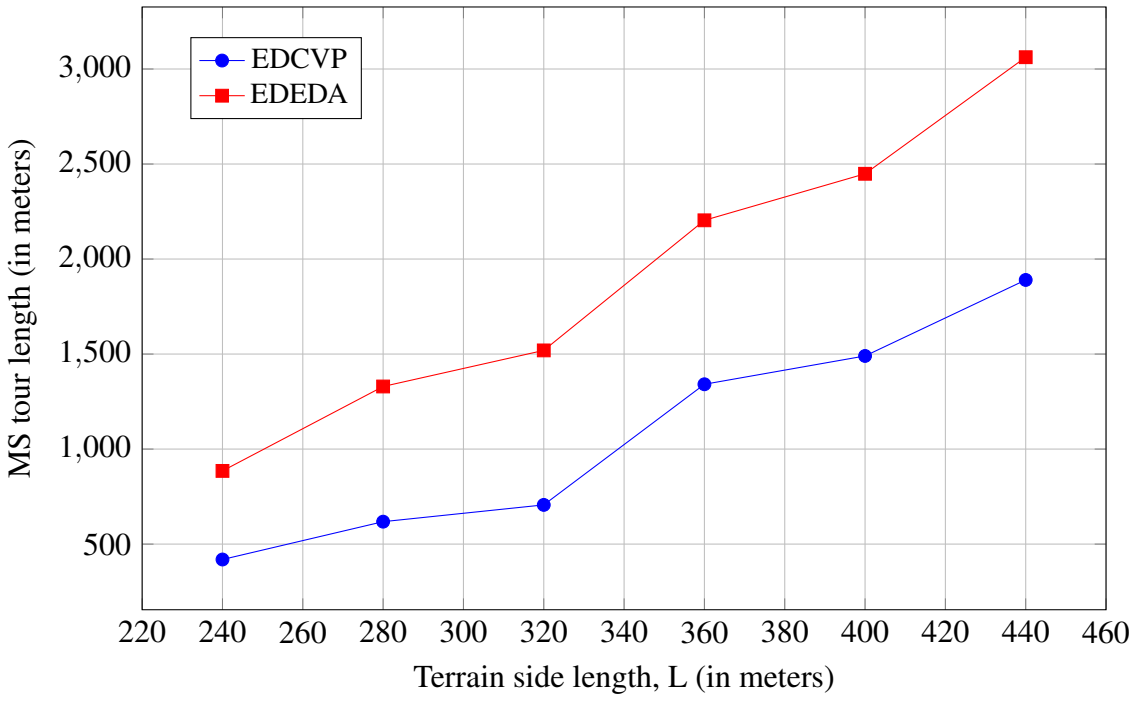


Figure 5.5: Comparison of EDCVP and EDEDA: Terrain side length (L) vs MS Tour Length.

Table 5.2: MS tour length, data collection delay, and number of VPs as a function of α for EDCVP and EDEDA protocols, along with the percentage reductions achieved.

R_{max}	**L**	α	**EDCVP**				**EDEDA**				**Tour Reduction (%)**	**Delay Reduction (%)**
			$k \times k$	#VPs	Tour length (m)	Delay (min)	$k \times k$	#VPs	Tour length (m)	Delay (min)		
80	240	8	9×9	4	418.885	1.36482	9×9	9	885.16	2.97527	111.26	118.01
80	280	9	9×9	4	617.99	1.69665	9×9	9	1329.35	4.88225	115.09	187.76
80	320	11	12×12	4	706.274	1.84378	12×12	16	1519.26	5.19877	115.17	181.94
80	360	12	12×12	4	1341.02	3.73503	12×12	16	2203.80	7.83967	64.35	109.93
80	400	14	15×15	9	1490.03	3.98338	15×15	25	2448.67	8.24778	64.36	107.02
80	440	15	15×15	9	1890.00	4.65	15×15	25	3062.10	11.1035	62.09	138.77

5.5 Conclusion

In this chapter, we validated our EDCVP protocol through simulations carried out with the NS-3 simulator. After presenting the simulation environment and the applied methodology, we evaluated the obtained results based on key performance metrics. The results are presented in tables and figures that enable clear and unbiased comparisons. It was found that EDCVP outperformed EDEDA in terms of energy consumption , network lifetime, MS tour length, and data collection delay. These enhancements demonstrate the effectiveness and suitability of our protocol for delay-constrained and energy-efficient data acquisition in WSNs.

Chapter 6

General Conclusion

6.1 Contributions

In this study, we addressed the problem of optimizing energy efficiency and data-collecting delays in WSNs by using a mobile sink-based protocol. Through the virtual grid segmentation of the sensing region, the proposed protocol, *Enhanced Data Collection via Visiting Points* (EDCVP), presents an organized way of data collection. Every grid selects a GCH based on residual energy and proximity to the grid center, ensuring effective local data gathering. A subset of these GCHs are RGCHs that gather data from neighboring GCHs and forward it to the MS. The MS collects data from strategically selected VPs, which are placed at the intersections of four adjacent grid cells, minimizing data redundancy and reducing communication overhead. The movement of the sink is represented as a Hamiltonian cycle and a backtracking method is employed as the means of finding this optimal route, thereby guaranteeing that all VPs are visited only once, reducing travel distance and hence collection delay. Re-election of GCHs based on their residual energy levels guarantees the network stays balanced and increases its operational lifetime. To evaluate the performance of our proposed EDCVP protocol, we implemented it in the NS-3 simulator. For comparison, we also implemented the EDEDA protocol, which similarly employs a virtual grid and an MS for data collection. Both protocols were simulated under identical conditions to ensure a fair comparison and to assess the improvements introduced by EDCVP. The simulation results demonstrate that EDCVP outperforms EDEDA by significantly reducing energy consumption and data collection delay and enhancing overall

network lifetime.

6.2 Limitations

Despite the promising results demonstrated by the EDCVP protocol, several limitations remain that may affect its applicability in real-world scenarios:

- The current version of the protocol is designed to operate with only one MS. While effective for moderate-sized networks, this limitation may become challenging in larger-scale deployments or in situations with high data-generating rates. Depending only on one sink could cause data-collecting obstacles, higher delays, and unequal network resource use.
- A backtracking algorithm was employed to determine the trajectory of the MS. While it often provides satisfactory results, it does not guarantee an optimal solution. Consequently, alternative methods for determining Hamiltonian cycles may be explored to reduce data collection latency.
- Free-obstacle sensing zones were assumed during the conception of EDCVP. Nonetheless, in actual deployment, such may not hold true. For example, in a forest fire detection application, the monitored terrain is frequently heavily vegetated and may have obstacles such as hills or dense trees. These characteristics hinder drones from adhering to a fixed, predefined flight trajectory.

6.3 Future work and perspectives

To overcome the current limitations and expand on this work, several future directions are suggested:

- Future research could explore dynamically adjusting the MS's path based on real-time conditions like node energy levels, data traffic, or unexpected obstacles.
- A predetermined MS's path length may serve as an input parameter to meet time-sensitive requirements.
- Extending the model to support multiple MSs operating collaboratively for better scalability.

- Integrating AI models could help optimize decision processes like selecting VPs, determining MS's path, or predicting node failures.

References

[1] D. Kandris, C. Nakas, D. Vomvas, and G. Koulouras, "Applications of wireless sensor networks: An up-to-date survey," *Applied System Innovation*, vol. 3, no. 1, p. 14, 2020. [Online]. Available: https://doi.org/10.3390/asi3010014

[2] S. Latambale and S. Sirsikar, "A survey of various sink mobility based techniques in wireless sensor network," in *Proceedings of the ACM Symposium on Women in Research 2016*, 2016, pp. 45–50. [Online]. Available: https://doi.org/10.1145/2909067.2909075

[3] "Ns-3, site officiel," 2025, [Online; accessed April 20, 2025]. [Online]. Available: https://www.nsnam.org/docs/tutorial/html/

[4] R. K. Verma and S. Jain, "Energy and delay efficient data acquisition in wireless sensor networks by selecting optimal visiting points for mobile sink," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 9, pp. 11 671–11 684, Sep. 2023. [Online]. Available: https://link.springer.com/10.1007/s12652-022-03729-9

[5] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, 2002. [Online]. Available: https://ieeexplore.ieee.org/document/1024422

[6] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008. [Online]. Available: https://doi.org/10.1016/j.comnet.2008.04.002

[7] R. Priyadarshi, B. Gupta, and A. Anurag, "Deployment techniques in wireless sensor networks: a survey, classification, challenges, and future research issues," *The Journal of Supercomputing*, vol. 76, pp. 7333–7373, 2020. [Online]. Available: https://doi.org/10.1007/s11227-020-03166-5

[8] L. Liu, F. Xia, Z. Wang, J. Chen, and Y. Sun, "Deployment issues in wireless sensor networks," in *Mobile Ad-hoc and Sensor Networks: First International Conference, MSN 2005, Wuhan, China, December 13-15, 2005. Proceedings 1*. Springer, 2005, pp. 239–248. [Online]. Available: https://doi.org/10.1007/11599463_24

[9] M. R. Senouci, A. Mellouk, and A. Aissani, "Random deployment of wireless sensor networks: a survey and approach," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 15, no. 1-3, pp. 133–146, 2014. [Online]. Available: https://www.inderscience.com/info/inarticle.php?artid=59905

[10] V. Sharma and A. Pughat, *Energy-Efficient Wireless Sensor Networks*. Boca Raton, FL: CRC Press, 2018, taylor & Francis Group. [Online]. Available: https://www.taylorfrancis.com/books/edit/10.1201/9781315155470/energy-efficient-wireless-sensor-networks-vidushi-sharma-anuradha-pughat.

[11] C. Nakas, D. Kandris, and G. Visvardis, "Energy efficient routing in wireless sensor networks: A comprehensive survey," *Algorithms*, vol. 13, no. 3, p. 72, March 2020, received: 28 February 2020; Accepted: 21 March 2020; Published: 24 March 2020. [Online]. Available: mailto:dkandris@uniwa.gr

[12] I. F. Akyildiz and M. C. Vuran, *Wireless Sensor Networks*, ser. Ian F. Akyildiz Series in Communications and Networking. Hoboken, NJ, USA: Wiley, 2010, graduate-level textbook in communications engineering and networking. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470515181

[13] S. H. A. Shalli Rani, "Multi-hop routing in wireless sensor networks: A survey," *IEEE Wireless Communications*, 2004. [Online]. Available: https://link.springer.com/book/10.1007/978-981-287-730-7

[14] S. M. Chowdhury and A. Hossain, "Different energy saving schemes in wireless sensor networks: A survey," *Wireless Personal Communications*, vol. 112, no. 1, 2020. [Online]. Available: https://doi.org/10.1007/s11277-020-07461-5

[15] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad hoc networks*, vol. 7, no. 3, pp. 537–568, 2009. [Online]. Available: https://doi.org/10.1016/j.adhoc.2008.06.003

[16] T. S. Rappaport, *Wireless communications: principles and practice*. New York: Cambridge University Press, 2024, includes bibliographical references and index. [Online]. Available: https://www.cambridge.org/

[17] J. G. Proakis and M. Salehi, *Digital communications*. New York: McGraw-hill, 2008, single-volume textbook on digital communications.

[18] O. J. Odeyinka, O. A. Ajibola, M. C. Ndinechi, O. C. Nosiri, and N. C. Onuekwusi, "A review on conservation of energy in wireless sensor networks," *International Journal of Smart Sensor Technologies and applications (IJSSTA)*, vol. 1, no. 2, pp. 1–16, 2020. [Online]. Available: https://orcid.org/0000-0003-0353-8431

[19] J. AL-KARAKI, "Routing techniques in wireless sensor networks: a survey," *IEEE Communication Magazine google schola*, vol. 2, 2004. [Online]. Available: https://ieeexplore.ieee.org/document/1368893

[20] S. Muruganathan, D. Ma, R. Bhasin, and A. Fapojuwo, "A centralized energy-efficient routing protocol for wireless sensor networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 8–13, 2005. [Online]. Available: https://doi.org/10.1109/MCOM.2005.1404592

[21] W. Zhang, G. Han, Y. Feng, and J. Lloret, "Irpl: An energy efficient routing protocol for wireless sensor networks," *Journal of Systems Architecture*, vol. 75, pp. 35–49, 2017. [Online]. Available: https://doi.org/10.1016/j.sysarc.2017.03.006

[22] A. Ali, Y. Ming, S. Chakraborty, and S. Iram, "A comprehensive survey on real-time applications of wsn," *Future internet*, vol. 9, no. 4, p. 77, 2017. [Online]. Available: https://doi.org/10.3390/fi9040077

[23] P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *The Journal of supercomputing*, vol. 68, pp. 1–48, 2014. [Online]. Available: https://hal.science/hal-00955283

[24] S. Prasanna and S. Rao, "An overview of wireless sensor networks applications and security," *International Journal of Soft Computing and Engineering (IJSCE)*, May 2012. [Online]. Available: https://www.ijsce.org/wp-content/uploads/papers/v2i2/B0648042212.pdf

[25] N. Temene, C. Sergiou, C. Georgiou, and V. Vassiliou, "A survey on mobility in wireless sensor networks," *Ad Hoc Networks*, vol. 125, p. 102726, 2022. [Online]. Available: https://doi.org/10.1016/j.adhoc.2021.102726

[26] S. A. Munir, B. Ren, W. Jiao, B. Wang, D. Xie, and J. Ma, "Mobile wireless sensor network: Architecture and enabling technologies for ubiquitous computing," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, vol. 2. IEEE, 2007, pp. 113–120. [Online]. Available: https://doi.org/10.1109/AINAW.2007.257

[27] R. Hamidouche, Z. Aliouat, A. M. Gueroui, A. A. A. Ari, and L. Louail, "Classical and bio-inspired mobility in sensor networks for iot applications," *Journal of Network and Computer Applications*, vol. 121, pp. 70–88, 2018. [Online]. Available: https://www.elsevier.com/locate/jnca

[28] A. Erman-Tüysüz, "Multi-sink mobile wireless sensor networks: dissemination protocols, design and evaluation," 2011. [Online]. Available: https://doi.org/10.3990/1.9789036532518

[29] Y.-G. Yue and P. He, "A comprehensive survey on the reliability of mobile wireless sensor networks: Taxonomy, challenges, and future directions," *Information Fusion*, vol. 44, pp. 188–204, 2018. [Online]. Available: https://doi.org/10.1016/j.inffus.2018.02.004

[30] B. Kechar and L. Sekhri, "Formal modelling and validation of a novel energy efficient cross-layer mac protocol in wireless multi hop sensor networks using time petri nets," in *2008 New Technologies, Mobility and Security*. Tangier, Morocco: IEEE, 2008, pp. 1–5. [Online]. Available: https://doi.org/10.1109/NTMS.2008.ECP.47

[31] Y. Derdour, B. Kechar, and F. Khelfi, "The impact of the mobile element on performance improvement in wireless sensor network," *Procedia Computer Science*, vol. 32, pp. 261–268, 2014. [Online]. Available: https://doi.org/10.1016/j.procs.2014.05.423

[32] R. Kacimi, R. Dhaou, and A.-L. Beylot, "Load balancing techniques for lifetime maximizing in wireless sensor networks," *Ad hoc networks*, vol. 11, no. 8, pp. 2172–2186, 2013. [Online]. Available: https://doi.org/10.1016/j.adhoc.2013.04.009

[33] E. Ahmed, I. Yaqoob, I. A. T. Hashem, I. Khan, A. I. A. Ahmed, M. Imran, and A. V. Vasilakos, "The role of big data analytics in internet of things," *Computer Networks*, vol. 129, pp. 459–471, 2017. [Online]. Available: https://doi.org/10.1016/j.comnet.2017.06.013

[34] U. A. Chude-Okonkwo, R. Malekian, B. T. Maharaj, and C. C. Chude, "Bio-inspired approach for eliminating redundant nanodevices in internet of bio-nano things," in *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/7414163

[35] R. Silva, J. S. Silva, and F. Boavida, "Mobility in wireless sensor networks–survey and proposal," *Computer Communications*, vol. 52, pp. 1–20, 2014. [Online]. Available: https://doi.org/10.1016/j.comcom.2014.05.008

[36] D. H. Stolfi, M. R. Brust, G. Danoy, and P. Bouvry, "Uav-ugv-umv multi-swarms for cooperative surveillance," *Frontiers in Robotics and AI*, vol. 8, p. 616950, 2021. [Online]. Available: https://doi.org/10.3389/frobt.2021.616950

[37] N. Sabor, S. Sasaki, M. Abo-Zahhad, and S. M. Ahmed, "A comprehensive survey on hierarchical-based routing protocols for mobile wireless sensor networks: review, taxonomy, and future directions," *Wireless Communications and Mobile Computing*, vol. 2017, no. 1, p. 2818542, 2017. [Online]. Available: https://www.hindawi.com/journals/wcmc/2017/2818542/

[38] A. W. Khan, A. H. Abdullah, M. H. Anisi, and J. I. Bangash, "A comprehensive study of data collection schemes using mobile sinks in wireless sensor networks," *Sensors*, vol. 14, no. 2, pp. 2510–2548, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/2/2510

[39] J.-Y. Chang, J.-T. Jeng, Y.-H. Sheu, Z.-J. Jian, and W.-Y. Chang, "An efficient data collection path planning scheme for wireless sensor networks with mobile sinks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, no. 1, p. 257, 2020. [Online]. Available: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-020-01873-4

[40] M. P. K, "Determination of energy efficient optimal sojourn location of mobile

sinks in clustered corona-based wireless sensor networks," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 1–12, Jan. 2022. [Online]. Available: https://link.springer.com/10.1007/s12083-021-01224-0

[41] A. Mehto, S. Tapaswi, and K. Pattanaik, "Virtual grid-based rendezvous point and sojourn location selection for energy and delay efficient data acquisition in wireless sensor networks with mobile sink," *Wireless Networks*, vol. 26, pp. 3763–3779, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11276-020-02293-4

[42] A. Mehto, R. K. Verma, and S. Jain, "Efficient Trajectory Planning and Route Adjustment Strategy for Mobile Sink in WSN-assisted IoT," in *2022 IEEE Region 10 Symposium (TENSYMP)*. Mumbai, India: IEEE, Jul. 2022, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9864434/

[43] W. Wen, S. Zhao, C. Shang, and C.-Y. Chang, "EAPC: Energy-Aware Path Construction for Data Collection Using Mobile Sink in Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 18, no. 2, pp. 890–901, Jan. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8106773/

[44] X. Fu and X. He, "Energy-balanced data collection with path-constrained mobile sink in wireless sensor networks," *AEU - International Journal of Electronics and Communications*, vol. 127, p. 153504, Dec. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1434841120319695

[45] Y. M. Raghavendra and U. B. Mahadevaswamy, "Hybrid Rendezvous Clustering Model for Efficient Data Collection in Multi Sink Based Wireless Sensor Networks," *Wireless Personal Communications*, vol. 129, no. 2, pp. 837–851, Mar. 2023. [Online]. Available: https://link.springer.com/10.1007/s11277-022-10158-6

[46] B. G. Gutam, P. K. Donta, C. S. R. Annavarapu, and Y.-C. Hu, "Optimal rendezvous points selection and mobile sink trajectory construction for data collection in WSNs," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 6, pp. 7147–7158, Jun. 2023. [Online]. Available: https://link.springer.com/10.1007/s12652-021-03566-2

[47] B. R. Al-Kaseem, Z. K. Taha, S. W. Abdulmajeed, and H. S. Al-Raweshidy, "Optimized Energy – Efficient Path Planning Strategy in WSN With Multiple

Mobile Sinks," *IEEE Access*, vol. 9, pp. 82 833–82 847, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9448016/

[48] R. Anwit, A. Tomar, and P. K. Jana, "Scheme for tour planning of mobile sink in wireless sensor networks," *IET Communications*, vol. 14, no. 3, pp. 430–439, Feb. 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1049/iet-com.2019.0613

[49] F. Engmann, F. A. Katsriku, J.-D. Abdulai, K. S. Adu-Manu, and F. K. Banaseka, "Prolonging the lifetime of wireless sensor networks: A review of current techniques," *Wireless Communications and Mobile Computing*, vol. 2018, no. 1, 2018. [Online]. Available: https://doi.org/10.1155/2018/8035065

Appendix A

EDCVP Protocol Implementation Files in NS-3

In this appendix, we present the main C++ source files used to implement the EDCVP protocol within the NS-3 simulation environment. These files contain the essential classes, functions, and configurations necessary for modeling SN behavior, sink mobility, data collection mechanisms, topology generation, and simulation setup. The goal of this appendix is to clarify the structure of the main code used for simulating the proposed EDCVP protocol, namely:

- **EDCVP simulation setup** (`edcvp.cc`): This is the main simulation file. It sets up the WSN by creating nodes (sensors and sink), assigning mobility models (fixed for sensors and a custom path for the sink), configuring simulation parameters, defining VPs, and launching the simulation.
- **Definition of EDCVP messages** (`edcvp_packet.h`): Defines the types of messages used in the EDCVP protocol. Each message includes a type header to identify the message purpose and a control header including essential control information.
- **Definition of the neighbors table** (`Neighbors_table.h`): Implements the neighbor table for each node. It stores details about nearby nodes within the same cell.
- **Definition of the EDCVP routing protocol logic** (`edcvp_protocol.h`): Defines all routing logic and behaviors including GCH election, data collection, and energy management.
- **Definition of the fixed path mobility model for sink** (`Fixed_path_mobility_model.h`): Implements the custom sink mobility model. It enables the MS to follow a predefined Hamiltonian cycle across selected VPs.

- **Definition of some essential utility functions for simulation** (`Helper_functions.h`): Provides utility functions essential for simulation, including topology generation, grid drawing, Hamiltonian cycle calculation, and connectivity checks.

- **Declaration of helper class for EDCVP routing protocol** (`edcvp_helper.h`): A helper class used to instantiate and install the EDCVP routing protocol on each node. It ensures that all necessary parameters are passed correctly during the initialization of the simulation environment.

A.1 EDCVP simulation setup

```
// ...
const int SIMULATION_DURATION = 60 * 60 * 96; // Simulation duration set to 96 hours (in seconds)
const int SENSORS_NUMBER = 300; // Number of sensor nodes in the network
const int SINK_NUMBER = 1; // Number of mobile sinks (typically one)
const int COMMUNICATION_RANGE = 80; // Communication range of sensor nodes (m)
const double PI = 3.14159265359; // Mathematical constant for calculations
const double L = 240;// Side length of the square simulation area (m)
const double MIN_DISTANCE = 10; // Minimum distance between nodes (m)
const double INIT_ENERGY = 1; // Initial energy for sensor nodes (J)
const double ENERGY_THRESHOLD = 0.4; // Energy threshold for node operation
const int SINK_CYCLES_NUMBER = 90; // Number of cycles for the sink's movement
const int INTERVAL_BETWEEN_TWO_CYCLES = 1000 * 60 * 60; // Interval between sink cycles: 60
    minutes (in milliseconds)
const int TIME_TO_CHECK_ENERGY = 1; // Time interval to check energy levels (in S)

// Global variables for simulation
double a; // Grid cell size
double cycleLength = 0; // Length of the sink's Hamiltonian cycle
int numberOfVisitingPoints = 0; // Number of visiting points for the sink

// Define the NS-3 logging component for debugging
NS_LOG_COMPONENT_DEFINE("WirelessSensorNetworkEDCVP");

// Function to collect and save simulation results (energy consumption, network lifetime, sink
    tour delay, and tour length)
void GetSimulationResults(NodeContainer& nodes) {
    // Vector to store node IDs and their times of death
    std::vector<std::pair<int, double>> instantsOfDeath;
    // Total energy consumed by all sensors
    double consumedEnergy = 0;
    // Energy consumed by data transmission
    double consumedEnergyByDaya = 0;

    // Iterate through sensor nodes (excluding the sink) to collect energy data
    for (NodeContainer::Iterator iter = nodes.Begin() + 1; iter != nodes.End(); ++iter) {
```

```
34          Ptr<Node> node = *iter;
35          // Calculate energy consumed by this node (initial energy minus remaining energy, capped
    at 0)
36          consumedEnergy = consumedEnergy + (INIT_ENERGY - std::max(node->GetObject<ns3::edcvp::
    EdcvpRoutingProtocol>()->GetRemainingEnergy(), 0.0));
37          // Accumulate energy consumed by data transmission
38          consumedEnergyByDaya = consumedEnergyByDaya + node->GetObject<ns3::edcvp::
    EdcvpRoutingProtocol>()->getTotalEnergyConsumedByDataTransmission();
39          // Store node ID and its time of death
40          instantsOfDeath.push_back(std::make_pair(node->GetId(), node->GetObject<ns3::edcvp::
    EdcvpRoutingProtocol>()->GetInstantOfDeath()));
41      }
42
43      // Find the earliest time of death to determine network lifetime
44      double minInstant = std::numeric_limits<double>::max();
45      for (const auto& pair : instantsOfDeath) {
46          if (pair.second < minInstant) {
47              minInstant = pair.second;
48          }
49      }
50
51      // Save total energy consumption to file
52      std::ofstream outFile1("./scratch/EDCVP_V6/energy_consumption.txt", std::ofstream::app);
53      if (!outFile1.is_open()) {
54          std::cerr << "Error: Unable to open energy_consumption.txt for writing" << std::endl;
55          return;
56      }
57      outFile1 << SENSORS_NUMBER << " " << consumedEnergy << std::endl;
58      outFile1.close();
59      // Print energy consumption results
60      std::cout << "Total energy consumption of EDCVP (Joules): " << consumedEnergy << std::endl;
61      std::cout << "Total energy consumption by asking data and data transmissions (Joules): " <<
    consumedEnergyByDaya << std::endl;
62      std::cout << "Total energy consumption by control messages (Joules): " << (consumedEnergy -
    consumedEnergyByDaya) << std::endl;
63
64      // Save network lifetime to file (converted to hours)
65      std::ofstream outFile2("./scratch/EDCVP_V6/lifetime.txt", std::ofstream::app);
66      if (!outFile2.is_open()) {
67          std::cerr << "Error: Unable to open lifetime.txt for writing" << std::endl;
68          return;
69      }
70      outFile2 << SENSORS_NUMBER << " " << minInstant / 3600 << std::endl;
71      outFile2.close();
72      std::cout << "Network lifetime (hours): " << minInstant / 3600 << std::endl;
73
74      // Save sink tour delay to file (converted to minutes)
75      std::ofstream outFile3("./scratch/EDCVP_V6/tourDelay.txt", std::ofstream::app);
76      if (!outFile3.is_open()) {
```

```
77          std::cerr << "Error: Unable to open file for writing" << std::endl;
78          return;
79      }
80      Ptr<FixedPathdMobilityModel> sinkMobility = nodes.Get(0)->GetObject<FixedPathdMobilityModel>()
        ;
81      outFile3 << L << " " << sinkMobility->GetSinkTourDelay() / (1000 * 60) << std::endl;
82      outFile3.close();
83      std::cout << "Sink tour delay (minutes): " << sinkMobility->GetSinkTourDelay() / (1000 * 60)
        << std::endl;
84      std::cout << "Number of visiting points: " << numberOfVisitingPoints << std::endl;
85
86      // Save sink tour length to file
87      std::ofstream outFile4("./scratch/EDCVP_V6/tourLength.txt", std::ofstream::app);
88      if (!outFile4.is_open()) {
89          std::cerr << "Error: Unable to open file for writing" << std::endl;
90          return;
91      }
92      outFile4 << L << " " << cycleLength << std::endl;
93      outFile4.close();
94      std::cout << "the MS tour length (meter): " << cycleLength << std::endl;
95  }
96
97  // Utility function to check if a value exists in a map of vectors
98  bool valueExistsInMap(const std::map<std::pair<int, int>, std::vector<int>>& myMap, int
        valueToFind) {
99      // Iterate through the map and check if the value exists in any vector
100     for (const auto& [key, vec] : myMap) {
101         if (std::find(vec.begin(), vec.end(), valueToFind) != vec.end()) {
102             return true;
103         }
104     }
105     return false;
106 }
107
108 // Main function to set up and run the EDCVP simulation
109 int main(int argc, char *argv[]) {
110     // Parse command-line arguments
111     CommandLine cmd;
112     cmd.Parse(argc, argv);
113
114     // Set WiFi physical layer mode to 1 Mbps DSSS
115     std::string phyMode("DsssRate1Mbps");
116     // Disable verbose logging by default
117     bool verbose = false;
118
119     // Create nodes for sensors and the sink
120     NodeContainer nodes;
121     MobilityHelper sensorMobility;
122     MobilityHelper sinkMobility;
```

```
123     nodes.Create(SENSORS_NUMBER + SINK_NUMBER);
124
125     // Calculate grid cell size 'a' based on communication range and area size
126     double alpha = ceil((2 * sqrt(2) * L) / COMMUNICATION_RANGE);
127     double k = alpha;
128     // Adjust grid size 'k' and cell size 'a' based on the computed alpha
129     if (alpha <= 6) {
130         a = L / 6;
131         k = 6;
132     } else if (alpha <= 9) {
133         a = L / 9;
134         k = 9;
135     } else if (alpha <= 12) {
136         a = L / 12;
137         k = 12;
138     } else if (alpha <= 15) {
139         a = L / 15;
140         k = 15;
141     } else if (alpha <= 18) {
142         a = L / 18;
143         k = 18;
144     }
145     std::cout << "K: " << k << " L: " << L << " a: " << a << std::endl;
146
147     // Set mobility model for the sink using FixedPathdMobilityModel for a predefined path
148     sinkMobility.SetMobilityModel("ns3::FixedPathdMobilityModel");
149     sinkMobility.Install(nodes.Get(0));
150     // Position the sink at the bottom center of the simulation area initially
151     nodes.Get(0)->GetObject<MobilityModel>()->SetPosition(Vector(L / 2, 0.0, 0.0));
152
153     // Generate random topology for sensor nodes, ensuring minimum distance constraints
154     GenerateTopology(sensorMobility, sinkMobility, nodes, L, COMMUNICATION_RANGE, MIN_DISTANCE);
155
156     // Identify visiting points for the sink's path
157     std::vector<VisitingPoint> visitingPoints;
158     // Add starting position at the bottom center
159     visitingPoints.push_back(VisitingPoint{L / 2, 0});
160
161     // Generate visiting points on a grid based on cell size 'a', skipping every other cell
162     int i = 0;
163     while (i <= (k / 3 - 2)) {
164         int j = 0;
165         double x = a * (3 * i + 3);
166         while (j <= (k / 3 - 2)) {
167             double y = a * (3 * j + 3);
168             visitingPoints.push_back(VisitingPoint{x, y});
169             numberOfVisitingPoints++;
170             // Skip every other cell in the row, except at the boundary
171             if (j + 1 == (k / 3 - 2)) {
```

```
172                 j = j + 1;
173             } else {
174                 j = j + 2;
175             }
176         }
177         // Skip every other row, except at the boundary
178         if (i + 1 == (k / 3 - 2)) {
179             i = i + 1;
180         } else {
181             i = i + 2;
182         }
183     }
184     // Print the coordinates of visiting points
185     std::cout << "\nVisiting Points:" << std::endl;
186     for (const auto& vp : visitingPoints) {
187         std::cout << "(" << vp.x << ", " << vp.y << ")" << std::endl;
188     }
189 
190     // Identify rendezvous cells (cells containing cluster heads)
191     std::vector<uint32_t> rCells;
192     for (int i = 0; i <= (k / 3 - 1); i++) {
193         int xx = 3 * i + 1;
194         for (int j = 0; j <= (k / 3 - 1); j++) {
195             int yy = 3 * j + 1;
196             uint32_t idOfCHCell = (yy) * k + xx;
197             rCells.push_back(idOfCHCell);
198         }
199     }
200     // Print rendezvous cell IDs
201     std::cout << "\nRendezvous cells:" << std::endl;
202     for (size_t i = 0; i < rCells.size(); ++i) {
203         std::cout << rCells[i];
204         if (i != rCells.size() - 1) {
205             std::cout << ",";
206         }
207     }
208     std::cout << "\n" << std::endl;
209 
210     // Define the sink's path using a Hamiltonian cycle
211     std::vector<std::vector<bool>> adjacencyMatrix = createAdjacencyMatrix(visitingPoints, a, k);
212 
213     // Display the adjacency matrix for debugging
214     i = 0;
215     for (const auto& row : adjacencyMatrix) {
216         for (bool value : row) {
217             std::cout << value << " ";
218         }
219         std::cout << "X: " << visitingPoints[i].x << " Y: " << visitingPoints[i].y << std::endl;
220         i++;
```

```
    }
    // Calculate the Hamiltonian cycle for the sink's path
    std::vector<VisitingPoint> hamiltonianCycle;
    hamCycle(adjacencyMatrix, visitingPoints, hamiltonianCycle);

    // Display the Hamiltonian cycle points
    std::cout << "\nHamiltonian cycle:" << std::endl;
    for (const auto& point : hamiltonianCycle) {
        std::cout << "(" << point.x << ", " << point.y << ")" << std::endl;
    }
    // Compute and print the total length of the Hamiltonian cycle
    cycleLength = ComputeCycleLength(hamiltonianCycle);
    std::cout << "Cycle length: " << cycleLength << std::endl;

    // Generate an SVG image of the grid with visiting points and rendezvous cells
    std::string gridFilename = "./scratch/EDCVP_V6/grid.svg";
    GenerateGridImage(gridFilename, k, a, visitingPoints, rCells);

    // Set the sink's mobility path using the Hamiltonian cycle
    Ptr<FixedPathdMobilityModel> sMobility = nodes.Get(0)->GetObject<FixedPathdMobilityModel>();
    std::vector<Vector> path;
    for (const auto& point : hamiltonianCycle) {
        path.push_back(Vector(point.x, point.y, 0.0));
    }
    sMobility->SetPath(path);
    sMobility->SetCycleNumber(SINK_CYCLES_NUMBER);
    sMobility->SetIntervalBetweenTwoCycles(INTERVAL_BETWEEN_TWO_CYCLES);

    // Initialize NetAnim for visualization, creating an XML file for animation
    AnimationInterface anim("edcvp.xml");

    // Prepare node parameters for the EDCVP routing protocol
    std::vector<std::tuple<
        uint32_t,           // Node ID
        double,             // X-coordinate
        double,             // Y-coordinate
        uint32_t,           // Sink node ID
        uint32_t,           // Grid cell ID
        uint32_t,           // Grid cell row
        uint32_t,           // Grid cell column
        uint32_t,           // Communication range
        uint32_t,           // Grid size parameter (k)
        double,             // X-coordinate of cell center
        double,             // Y-coordinate of cell center
        double,             // Initial energy
        double,             // Grid cell size (a)
        double,             // Energy threshold
        uint32_t,           // Time interval to check energy
        bool,               // Flag indicating if node is in a rendezvous cell
```

```
270         std::map<std::pair<int, int>, std::vector<int>>*, // List of cluster heads for data
    requests
271         AnimationInterface* // NetAnim interface for visualization
272     >> nodeParams;
273     // Map to store cluster heads for data requests at each visiting point
274     std::map<std::pair<int, int>, std::vector<int>> GChsToAskDataFromList;
275     // ID of the sink node
276     uint32_t sinkId = 0;
277     // Configure parameters for each node
278     for (uint32_t i = 0; i < nodes.GetN(); ++i) {
279         uint32_t nodeId = i;
280         // Get node position
281         double myX = nodes.Get(i)->GetObject<MobilityModel>()->GetPosition().x;
282         double myY = nodes.Get(i)->GetObject<MobilityModel>()->GetPosition().y;
283         // Calculate grid cell coordinates
284         uint32_t p = floor(myY / a);
285         uint32_t q = floor(myX / a);
286         uint32_t idCell = p * k + q;
287         uint32_t rowCell = floor(myY / a);
288         uint32_t colCell = floor(myX / a);
289         // Calculate cell center coordinates
290         double xCellCenter = (colCell + 1) * a - (a / 2);
291         double yCellCenter = (rowCell + 1) * a - (a / 2);
292         // Check if the node is in a rendezvous cell
293         bool ifIAmInRC = std::find(rCells.begin(), rCells.end(), idCell) != rCells.end();
294         // For the sink node, identify cluster heads to collect data from at each visiting point
295         if (i == 0) {
296             auto it = hamiltonianCycle.begin();
297             ++it; // Skip the starting point
298             for (; it != hamiltonianCycle.end(); ++it) {
299                 VisitingPoint point = *it;
300                 // Identify surrounding cells for data collection
301                 p = floor((point.y - a * 1.5) / a);
302                 q = floor((point.x - a * 1.5) / a);
303                 int idOfCHCell = (p * k) + q;
304                 if (!valueExistsInMap(GChsToAskDataFromList, idOfCHCell))
305                     GChsToAskDataFromList[{point.x, point.y}].push_back(idOfCHCell);
306 
307                 p = floor((point.y - a * 1.5) / a);
308                 q = floor((point.x + a * 1.5) / a);
309                 int idOfCHCell = (p * k) + q;
310                 if (!valueExistsInMap(GChsToAskDataFromList, idOfCHCell))
311                     GChsToAskDataFromList[{point.x, point.y}].push_back(idOfCHCell);
312 
313                 p = floor((point.y + a * 1.5) / a);
314                 q = floor((point.x - a * 1.5) / a);
315                 int idOfCHCell = (p * k) + q;
316                 if (!valueExistsInMap(GChsToAskDataFromList, idOfCHCell))
317                     GChsToAskDataFromList[{point.x, point.y}].push_back(idOfCHCell);
```

```
318
319                 p = floor((point.y + a * 1.5) / a);
320                 q = floor((point.x + a * 1.5) / a);
321                 int idOfCHCell = (p * k) + q;
322                 if (!valueExistsInMap(GChsToAskDataFromList, idOfCHCell))
323                     GChsToAskDataFromList[{point.x, point.y}].push_back(idOfCHCell);
324             }
325
326             // Print cluster heads for each visiting point
327             std::cout << "\nList RGCHs from whom the MS asks data from each visiting point:" <<
    std::endl;
328             for (const auto& entry : GChsToAskDataFromList) {
329                 std::cout << "\nVisiting Point: (" << entry.first.first << ", " << entry.first.
    second << ")\n";
330                 std::cout << "Ids RGCHs:";
331                 for (int value : entry.second) {
332                     std::cout << " " << value;
333                 }
334                 std::cout << std::endl;
335             }
336         }
337         // Store node parameters for the routing protocol
338         nodeParams.push_back(std::make_tuple(nodeId, myX, myY, sinkId, idCell, rowCell, colCell,
    COMMUNICATION_RANGE, k, xCellCenter, yCellCenter, INIT_ENERGY, a, ENERGY_THRESHOLD,
    TIME_TO_CHECK_ENERGY, ifIAmInRC, &GChsToAskDataFromList, &anim));
339     }
340     // Initialize the EDCVP routing protocol with node parameters
341     EdcvpHelper edcvp(nodeParams);
342     // Install the internet stack with the EDCVP routing protocol on all nodes
343     InternetStackHelper stack;
344     stack.SetRoutingHelper(edcvp);
345     stack.Install(nodes);
346     // Assign IP addresses to nodes starting from the base address
347     Ipv4AddressHelper address;
348     address.SetBase("10.1.0.0", "255.255.0.0");
349     // Configure point-to-point links for reliable communication
350     PointToPointHelper p2p;
351     p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
352     p2p.SetChannelAttribute("Delay", StringValue("2ms"));
353     // Establish point-to-point links between sensor nodes within communication range
354     // Point-to-point links ensure reliable communication without wireless uncertainties
355     for (uint32_t i = 1; i < nodes.GetN(); ++i) {
356         double Xi = nodes.Get(i)->GetObject<MobilityModel>()->GetPosition().x;
357         double Yi = nodes.Get(i)->GetObject<MobilityModel>()->GetPosition().y;
358         for (uint32_t j = 1; j < nodes.GetN(); ++j) {
359             if (i < j) {
360                 double Xj = nodes.Get(j)->GetObject<MobilityModel>()->GetPosition().x;
361                 double Yj = nodes.Get(j)->GetObject<MobilityModel>()->GetPosition().y;
362                 double distanceBetweenIJ = std::sqrt(std::pow(Xi - Xj, 2) + std::pow(Yi - Yj, 2));
```

```
                if (distanceBetweenIJ <= COMMUNICATION_RANGE) {
                    NetDeviceContainer devices = p2p.Install(nodes.Get(i), nodes.Get(j));
                    address.Assign(devices);
                }
            }
        }
    }

    // Connect the mobile sink to all sensor nodes using point-to-point links
    for (uint32_t i = 1; i < nodes.GetN(); ++i) {
        NetDeviceContainer devices = p2p.Install(nodes.Get(0), nodes.Get(i));
        address.Assign(devices);
    }

    // Configure NetAnim visualization for the sink node
    anim.UpdateNodeDescription(nodes.Get(0), "Sink"); // Label the sink node
    uint32_t resourceId = anim.AddResource("/home/nadjib/ns-allinone-3.41/ns-3.41/scratch/EDCVP_V6
    /drone.png");
    anim.UpdateNodeImage(nodes.Get(0)->GetId(), resourceId); // Set a custom image for the sink
    anim.UpdateNodeSize(nodes.Get(0), 30, 30); // Set size for the sink node
    // Set maximum packets for the NetAnim trace file
    anim.SetMaxPktsPerTraceFile(1000000);
    // Configure size and color for sensor nodes in visualization
    for (int i = 1; i <= SENSORS_NUMBER; ++i) {
        anim.UpdateNodeSize(nodes.Get(i), 3, 3); // Set size for sensor nodes
        anim.UpdateNodeColor(nodes.Get(i)->GetId(), 0, 0, 255); // Set blue color for sensor nodes
    }
    // Set the background image for NetAnim visualization
    std::string imagePath = "./scratch/EDCVP_V6/grid.svg";
    fs::path absolutePath = fs::absolute(imagePath);
    anim.SetBackgroundImage(absolutePath, 0, 0, 1, 1, 1);
    // Schedule the collection of simulation results just before the simulation ends
    Simulator::Schedule(Seconds(SIMULATION_DURATION - 2), &GetSimulationResults, nodes);
    // Set the simulation duration and run
    Simulator::Stop(Seconds(SIMULATION_DURATION));
    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

A.2 Definition of EDCVP messages

```
// ...
namespace ns3 {
namespace edcvp {

```

```
// Enumeration defining the types of messages used in the EDCVP protocol
enum MessageType {
    GCH_ELECTION,      // Message for electing Group Cluster Heads (GCH)
    GCH_ANNOUNCEMENT, // Message for announcing GCH status
    DATA_REQUEST,      // Message for requesting data from nodes
    DATA,              // Message containing sensor data
    GCH_REELECTION     // Message for reelecting GCHs based on energy levels
};
// Base header class to specify the type of EDCVP message
class TypeHeader : public Header {
public:
    // Constructor with default message type set to GCH_ELECTION
    TypeHeader(MessageType t = GCH_ELECTION);
    // NS-3 type system integration
    static TypeId GetTypeId();
    TypeId GetInstanceTypeId() const override;
    // Get the serialized size of the header
    uint32_t GetSerializedSize() const override;
    // Serialize the header into a buffer
    void Serialize(Buffer::Iterator i) const override;
    // Deserialize the header from a buffer
    uint32_t Deserialize(Buffer::Iterator start) override;
    // Print the header contents for debugging
    void Print(std::ostream& os) const override;
    // Equality operator to compare two TypeHeader instances
    bool operator==(const TypeHeader& o) const;
    // Stream output operator for printing the header
    friend std::ostream& operator<<(std::ostream& os, const TypeHeader& h);
    // Get the message type
    MessageType Get() const {
        return m_type;
    }

    // Check if the message type is valid
    bool IsValid() const {
        return m_valid;
    }

private:
    MessageType m_type; // The type of EDCVP message
    bool m_valid;       // Flag indicating if the message type is valid
};
// Header for GCH election messages, used to select cluster heads based on energy and position
class GCH_ElectionHeader : public Header {
public:
    // Constructor with default values for sender ID, cell ID, coordinates, remaining energy, and
    RGCH status
    GCH_ElectionHeader(uint32_t idSender = 0, uint32_t idCell = 0, double x = 0, double y = 0,
    double RE = 0, bool IAmRGCH = false);
```

```
52      // NS-3 type system integration
53      static TypeId GetTypeId();
54      TypeId GetInstanceTypeId() const override;
55
56      uint32_t GetSerializedSize() const override; // Get the serialized size of the header
57
58      void Serialize(Buffer::Iterator start) const override; // Serialize the header into a buffer
59
60      uint32_t Deserialize(Buffer::Iterator start) override; // Deserialize the header from a buffer
61
62      void Print(std::ostream& os) const override; // Print the header contents for debugging
63
64      void SetIdSender(uint32_t idSender) { m_idSender = idSender; }
65      uint32_t GetIdSen  // Getters and setters for header fields
66  der() const { return m_idSender; }
67      void SetIdCell(uint32_t idCell) { m_idCell = idCell; }
68      uint32_t GetIdCell() const { return m_idCell; }
69      void SetX(double x) { m_x = x; }
70      double GetX() const { return m_x; }
71      void SetY(double y) { m_y = y; }
72      double GetY() const { return m_y; }
73      void SetRE(double RE) { m_RE = RE; } // Set remaining energy
74      double GetRE() const { return m_RE; }
75      void SetIAMRGCH(bool IAmRGCH) { m_IAmRGCH = IAmRGCH; } // Set Rendezvous GCH status
76      bool GetIAMRGCH() const { return m_IAmRGCH; }
77      // Equality operator to compare two GCH_ElectionHeader instances
78      bool operator==(const GCH_ElectionHeader& o) const;
79
80  private:
81      uint32_t m_idSender; // ID of the sending node
82      uint32_t m_idCell;   // ID of the grid cell
83      double m_x, m_y;     // Coordinates of the sending node
84      double m_RE;         // Remaining energy of the sending node
85      bool m_IAmRGCH;      // Flag indicating if the sender is a Rendezvous GCH
86  };
87
88  // Header for GCH announcement messages, used to broadcast GCH status to neighbors
89  class GCH_AnnouncementHeader : public Header {
90  public:
91      // Constructor with default values for sender ID, cell ID, coordinates, and RGCH status
92      GCH_AnnouncementHeader(uint32_t idSender = 0, uint32_t idCell = 0, double x = 0, double y = 0,
         bool IAmRGCH = false);
93      // NS-3 type system integration
94      static TypeId GetTypeId();
95      TypeId GetInstanceTypeId() const override;
96
97      uint32_t GetSerializedSize() const override; // Get the serialized size of the header
98
99      void Serialize(Buffer::Iterator start) const override; // Serialize the header into a buffer
```

```
100
101     uint32_t Deserialize(Buffer::Iterator start) override; // Deserialize the header from a buffer
102
103     void Print(std::ostream& os) const override; // Print the header contents for debugging
104
105     // Getters and setters for header fields
106     void SetIdSender(uint32_t idSender) { m_idSender = idSender; }
107     uint32_t GetIdSender() const { return m_idSender; }
108     void SetIdCell(uint32_t idCell) { m_idCell = idCell; }
109     uint32_t GetIdCell() const { return m_idCell; }
110     void SetX(double x) { m_x = x; }
111     double GetX() const { return m_x; }
112     void SetY(double y) { m_y = y; }
113     double GetY() const { return m_y; }
114     void SetIAMRGCH(bool IAmRGCH) { m_IAmRGCH = IAmRGCH; } // Set Rendezvous GCH status
115     bool GetIAMRGCH() const { return m_IAmRGCH; }
116
117     // Equality operator to compare two GCH_AnnouncementHeader instances
118     bool operator==(const GCH_AnnouncementHeader& o) const;
119
120 private:
121     uint32_t m_idSender; // ID of the sending node
122     uint32_t m_idCell;   // ID of the grid cell
123     double m_x, m_y;     // Coordinates of the sending node
124     bool m_IAmRGCH;      // Flag indicating if the sender is a Rendezvous GCH
125 };
126
127 // Header for data request messages, sent by the sink to request data from cluster heads
128 class DATA_RequestHeader : public Header {
129 public:
130     // Constructor with default values for sender ID, cell ID, sink coordinates, and relevant
        RGCHs
131     DATA_RequestHeader(uint32_t idSender = 0, uint32_t idCell = 0, double xSink = 0, double ySink
        = 0, const std::vector<int>& relevantRGCHs = {});
132
133     // NS-3 type system integration
134     static TypeId GetTypeId();
135     TypeId GetInstanceTypeId() const override;
136
137     uint32_t GetSerializedSize() const override; // Get the serialized size of the header
138
139     void Serialize(Buffer::Iterator start) const override; // Serialize the header into a buffer
140
141     uint32_t Deserialize(Buffer::Iterator start) override; // Deserialize the header from a buffer
142
143     void Print(std::ostream& os) const override; // Print the header contents for debugging
144
145
146     // Getters and setters for header fields
```

```
147     void SetIdSender(uint32_t idSender) { m_idSender = idSender; }
148     uint32_t GetIdSender() const { return m_idSender; }
149 
150     void SetIdCell(uint32_t idCell) { m_idCell = idCell; }
151     uint32 Juno    uint32_t GetIdCell() const { return m_idCell; }
152     void SetRelevantRGCHs(std::vector<int>& relevantRGCHs) { m_relevantRGCHs = relevantRGCHs; }
153     const std::vector<int>& GetRelevantRGCHs() const { return m_relevantRGCHs; }
154     void SetXSink(double x) { m_xSink = x; }
155     double GetXSink() const { return m_xSink; }
156     void SetYSink(double y) { m_ySink = y; }
157     double GetYSink() const { return m_ySink; }
158 
159     // Equality operator to compare two DATA_RequestHeader instances
160     bool operator==(const DATA_RequestHeader& o) const;
161 
162 private:
163     uint32_t m_idSender;          // ID of the sending node (sink)
164     uint32_t m_idCell;            // ID of the grid cell
165     double m_xSink, m_ySink;      // Coordinates of the sink
166     std::vector<int> m_relevantRGCHs; // List of relevant Rendezvous GCHs to request data from
167 };
168 
169 // Header for data messages, containing sensor data sent to the sink
170 class DataHeader : public Header {
171 public:
172     // Constructor with default values for receiver, sender, cell ID, data, coordinates, and
    distance
173     DataHeader(uint32_t receiver = 0, uint32_t sender = 0, uint32_t cellIdSender = 0, const std::
    string& data = "",
174                double xSender = 0, double ySender = 0, double distanceToDestination = 0);
175 
176     // NS-3 type system integration
177     static TypeId GetTypeId();
178     TypeId GetInstanceTypeId() const override;
179 
180     uint32_t GetSerializedSize() const override; // Get the serialized size of the header
181 
182     void Serialize(Buffer::Iterator start) const override; // Serialize the header into a buffer
183 
184 
185     // Deserialize the header from a buffer
186     uint32_t Deserialize(Buffer::Iterator start) override;
187 
188     void Print(std::ostream& os) const override; // Print the header contents for debugging
189 
190     // Getters and setters for header fields
191     void SetReceiver(uint32_t receiver) { m_receiver = receiver; }
192     uint32_t GetReceiver() const { return m_receiver; }
193 
```

```
194     void SetSender(uint32_t sender) { m_sender = sender; }
195     uint32_t GetSender() const { return m_sender; }
196 
197     void SetCellIdSender(uint32_t cellIdSender) { m_cellId_Sender = cellIdSender; }
198     uint32_t GetCellIdSender() const { return m_cellId_Sender; }
199 
200     void SetXSender(double xSender) { m_xSender = xSender; }
201     double GetXSender() const { return m_xSender; }
202 
203     void SetYSender(double ySender) { m_ySender = ySender; }
204     double GetYSender() const { return m_ySender; }
205 
206     void SetDistanceToDestination(double distanceToDestination) { m_distanceToDestination =
        distanceToDestination; }
207     double GetDistanceToDestination() const { return m_distanceToDestination; }
208 
209     void SetData(const std::string& data) { m_data = data; }
210     std::string GetData() const { return m_data; }
211 
212     // Equality operator to compare two DataHeader instances
213     bool operator==(const DataHeader& o) const;
214 
215 private:
216     uint32_t m_receiver;           // ID of the receiving node (sink)
217     uint32_t m_sender;             // ID of the sending node
218     uint32_t m_cellId_Sender;      // ID of the sender's grid cell
219     std::string m_data;            // Sensor data payload
220     double m_distanceToDestination; // Distance to the destination (sink)
221     double m_xSender, m_ySender;  // Coordinates of the sending node
222 };
223 
224 // Header for GCH reelection messages, used to initiate reelection based on energy levels
225 class GCH_ReelectionHeader : public Header {
226 public:
227     // Constructor with default values for sender ID, cell ID, and current energy
228     GCH_ReelectionHeader(uint32_t idSender = 0, uint32_t idCell = 0, double m_currentEnergy = 0);
229 
230     // NS-3 type system integration
231     static TypeId GetTypeId();
232     TypeId GetInstanceTypeId() const override;
233 
234     uint32_t GetSerializedSize() const override; // Get the serialized size of the header
235 
236     void Serialize(Buffer::Iterator start) const override; // Serialize the header into a buffer
237 
238     uint32_t Deserialize(Buffer::Iterator start) override; // Deserialize the header from a buffer
239 
240     void Print(std::ostream& os) const override; // Print the header contents for debugging
241 
```

```
242     // Getters and setters for header fields
243     void SetIdSender(uint32_t idSender) { m_idSender = idSender; }
244     uint32_t GetIdSender() const { return m_idSender; }
245     void SetIdCell(uint32_t idCell) { m_idCell = idCell; }
246     uint32_t GetIdCell() const { return m_idCell; }
247     void SetCurrentEnergy(double energy) { m_currentEnergy = energy; }
248     double GetCurrentEnergy() const { return m_currentEnergy; }
249 
250     // Equality operator to compare two GCH_ReelectionHeader instances
251     bool operator==(const GCH_ReelectionHeader& o) const;
252 
253 private:
254     uint32_t m_idSender;       // ID of the sending node
255     uint32_t m_idCell;         // ID of the grid cell
256     double m_currentEnergy;    // Current energy level of the sending node
257 };
258 
259 } // namespace edcvp
260 } // namespace ns3
261 
262 #endif /* EDCVPPACKET_H */
```

A.3 Definition of the neighbors table

```
1  // ...
2  namespace ns3 {
3  namespace edcvp {
4  
5  // Class to manage a table of neighboring nodes for the EDCVP routing protocol
6  class Neighbors_table {
7  public:
8      // Structure to store information about a neighbor node
9      struct Neighbor {
10         int idNeighbor;    // Unique ID of the neighbor node
11         int idCell;        // ID of the grid cell the neighbor belongs to
12         double x;          // X-coordinate of the neighbor
13         double y;          // Y-coordinate of the neighbor
14         double RE;         // Remaining energy of the neighbor
15         bool IAMRGCH;      // Flag indicating if the neighbor is a Rendezvous Group Cluster Head (
       RGCH)
16 
17         // Constructor to initialize neighbor properties
18         Neighbor(int idN, int idC, double x, double y, double re, bool IAmRGCH)
19             : idNeighbor(idN), idCell(idC), x(x), y(y), RE(re), IAMRGCH(IAmRGCH)
20         {
21         }
22     };
```

```
23
24      // Constructor to initialize an empty neighbors table
25      Neighbors_table();
26
27      // Add a neighbor to the table with specified properties
28      void AddNeighbor(int idN, int idC, double x, double y, double re, bool IAmRGCH);
29
30      double GetRE(int idN); // Get the remaining energy of a neighbor by ID
31      double GetX(int idN); // Get the X-coordinate of a neighbor by ID
32      double GetY(int idN); // Get the Y-coordinate of a neighbor by ID
33      bool GetIAMRGCH(int idN); // Check if a neighbor is a Rendezvous GCH by ID
34      int CandidateNodesNumber(int cellId); // Get the number of candidate nodes in a specific cell
35      int MyNeighborsInCellNumber(int cellId); // Get the number of neighbors in a specific cell
36      bool NeighborExists(int idN); // Check if a neighbor with the given ID exists in the table
37
38      // Update the remaining energy and RGCH status of a neighbor
39      void UpdateRE(int idN, double re, bool IAmRGCH = false);
40
41      // Clear the neighbors table
42      void Clear();
43
44      // Vector storing the list of neighbors
45      std::vector<Neighbor> m_neighbors;
46  };
47
48  } // namespace edcvp
49  } // namespace ns3
50
51  #endif /* NEIGHBORS_TABLE_H */
```

A.4 Definition of the EDCVP routing protocol logic

```
1  namespace ns3 {
2  namespace edcvp {
3
4  // Routing protocol class for EDCVP, managing data collection in wireless sensor networks
5  class EdcvpRoutingProtocol : public Ipv4RoutingProtocol {
6  public:
7      static const uint32_t EDCVP_PORT; // Port number for EDCVP communication
8      // Register TypeId for NS-3 object system
9      static TypeId GetTypeId();
10
11     // Get the TypeId of this instance
12     TypeId GetInstanceTypeId() const override;
13
14     // Default constructor
15     EdcvpRoutingProtocol();
```

```
16
17      // Constructor with node-specific parameters
18      EdcvpRoutingProtocol(
19          uint32_t myId, // Node ID
20          double myX, // X-coordinate
21          double myY, // Y-coordinate
22          uint32_t sinkId, // Sink node ID
23          uint32_t idCell, // Grid cell ID
24          uint32_t rowCell, // Grid cell row
25          uint32_t colCell, // Grid cell column
26          uint32_t maxRange, // Maximum communication range
27          uint32_t k, // Grid size parameter
28          double xCellCenter, // X-coordinate of cell center
29          double yCellCenter, // Y-coordinate of cell center
30          double myEnergy, // Initial energy
31          double a, // Grid cell size
32          double energy_threshold, // Energy threshold for operation
33          uint32_t timeToCheckMyEnergy, // Time interval to check energy
34          bool IfIAmInRCell, // Flag for rendezvous cell
35          std::map<std::pair<int, int>, std::vector<int>>* GChsToAskDataFromList, // Cluster heads
    for data requests
36          AnimationInterface* anim // NetAnim interface for visualization
37      );
38      ~EdcvpRoutingProtocol();      // Destructor
39
40      void DoDispose() override;    // Clean up resources
41
42
43      // Determine route for outgoing packet
44      Ptr<Ipv4Route> RouteOutput(Ptr<Packet> p, const Ipv4Header& header, Ptr<NetDevice> oif, Socket
    ::SocketErrno& sockerr) override;
45
46      // Handle incoming packet routing
47      bool RouteInput(Ptr<const Packet> p, const Ipv4Header& header, Ptr<const NetDevice> idev,
48                      const UnicastForwardCallback& ucb, const MulticastForwardCallback& mcb,
49                      const LocalDeliverCallback& lcb, const ErrorCallback& ecb) override;
50
51      void NotifyInterfaceUp(uint32_t interface) override; // Handle interface activation
52
53      void NotifyInterfaceDown(uint32_t interface) override;// Handle interface deactivation
54
55      void NotifyAddAddress(uint32_t interface, Ipv4InterfaceAddress address) override; // Handle
    addition of IP address
56
57      void NotifyRemoveAddress(uint32_t interface, Ipv4InterfaceAddress address) override; // Handle
     removal of IP address
58
59      void SetIpv4(Ptr<Ipv4> ipv4) override; // Set IPv4 protocol instance
60
```

```
61      void PrintRoutingTable(Ptr<OutputStreamWrapper> stream, Time::Unit unit = Time::S) const
        override;  // Print routing table
62
63     void Start();  // Initialize protocol operations
64
65      void RecvEdcvp(Ptr<Socket> socket); // Receive EDCVP packet
66
67      void SendTo(Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination); // Send packet
        to destination
68
69      bool IsMyOwnAddress(Ipv4Address src); // Check if address is local
70
71      // Cluster Head Election
72      void SendGchElection(bool scheduleFunction);  // Send cluster head election message
73
74      void RecvGchElection(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src); // Receive cluster
         head election message
75
76      void IfIAMGCH(); // Check if this node is a cluster head
77
78
79      // Cluster Head Announcement
80      void SendGCH_ANNOUNCEMENT(); // Send cluster head announcement
81
82      void RecvGCH_ANNOUNCEMENT(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src); // Receive
        cluster head announcement
83
84      // Data Request and Sending
85      void CourseChange(std::string context, Ptr<const MobilityModel> mobility); // Handle mobility
        course change
86
87      // Send data request to cluster heads
88      void SendDataRequest(std::vector<int>& RGChsCellIds);
89
90      void SendDataRequest(); // Send data request (overload)
91
92      // Receive data request
93      void RecvDataRequest(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src);
94
95      void SendData(uint32_t receiverId); // Send data to receiver
96
97      void RecvData(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src);  // Receive data packet
98
99      // Cluster Head Reelection
100     void CheckEnergyForReelection(); // Check energy for reelection
101
102     void SendGchReelection();  // Send reelection message
103
104
```

```
105     void RecvGchReelection(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src); // Receive
        reelection message
106
107     double GetRemainingEnergy(); // Get remaining energy
108
109     double GetInstantOfDeath(); // Get time of node death
110
111     // Update energy for transmission
112     void updateEnergyTx(int size, double distance, bool data);
113
114     void updateEnergyRx(int size, bool data); // Update energy for reception
115
116     // Get total energy consumed by data transmission
117     double getTotalEnergyConsumedByDataTransmission();
118
119     int64_t AssignStreams(int64_t stream); // Assign random variable streams
120
121
122     // Find socket for interface address
123     Ptr<Socket> FindSocketWithInterfaceAddress(Ipv4InterfaceAddress iface) const;
124
125     // Member variables
126     uint32_t m_myId; // Node ID
127     double m_myX, m_myY; // Node coordinates
128     uint32_t m_sinkId; // Sink node ID
129     uint32_t m_idCell; // Grid cell ID
130     uint32_t m_rowCell, m_colCell; // Grid cell row and column
131     uint32_t m_maxRange; // Maximum communication range
132     uint32_t m_k; // Grid size parameter
133     double m_xCellCenter, m_yCellCenter; // Cell center coordinates
134     double m_myEnergy; // Current energy
135     double m_a; // Grid cell size
136     double m_energy_threshold; // Energy threshold for operation
137     double m_init_energy; // Initial energy
138     uint32_t m_timeToCheckMyEnergy; // Time interval to check energy
139     double m_instantOfDeath = std::numeric_limits<double>::max(); // Time of node death
140     bool m_IAmGCH = false, m_IAmRGCH = false; // Flags for cluster head status
141     bool m_IfIAmInRCell; // Flag for rendezvous cell
142     uint32_t m_idGCH = -1, m_idRGCH = -1; // IDs of cluster heads
143     double m_SentREnergy; // Sent remaining energy
144     double xSender, ySender; // Sender coordinates
145
146     Neighbors_table m_neighborsTable; // Table of neighboring nodes
147     Ptr<Ipv4> m_ipv4; // IPv4 protocol instance
148     AnimationInterface* m_anim; // NetAnim interface
149     std::map<std::pair<int, int>, std::vector<int>>* m_GChsDataRequestFromList; // Cluster heads
        for data requests
150     std::map<Ptr<Socket>, Ipv4InterfaceAddress> m_socketAddresses; // Socket to interface address
        mapping
```

```
151     std::map<Ptr<Socket>, Ipv4InterfaceAddress> m_socketSubnetBroadcastAddresses; // Socket to
        subnet broadcast mapping
152
153     Timer m_EMtimer; // Timer for election messages
154     Timer m_ifIAmGCHTimer; // Timer for cluster head check
155     Timer m_dataSendingTimer; // Timer for data sending
156     Timer m_CheckMyEnergyTimer; // Timer for energy checks
157     Timer m_gchAnnounceTimer; // Timer for cluster head announcements
158
159     uint32_t m_currentDataRound = 0; // Current data collection round
160     double totalEnergyConsumedByDataTransmission = 0; // Energy consumed by data transmission
161
162     bool firstCheck = false; // Flag for initial check
163     bool remainGCh = false; // Flag to remain cluster head
164     bool IMSent = false; // Flag for sent message
165     bool IDied = false; // Flag for node death
166
167     Ptr<UniformRandomVariable> m_uniformRandomVariable; // Random variable for uniform
        distribution
168     double d0 = 83.6; // Reference distance for energy model
169
170     double Efriss_amp = 100 * 1e-12; // Friss model energy (J/bit/m^2)
171     double Etwo_ray_amp = 0.013 * 1e-12; // Two-ray model energy (J/bit/m^4)
172     double EXcvr = 50 * 1e-9; // Transceiver energy (J/bit)
173
174 protected:
175     // Initialize protocol
176     virtual void DoInitialize() override;
177 };
178
179 } // namespace edcvp
180 } // namespace ns3
181
182 #endif // EDCVP_PROTOCOL_H
```

A.5 Definition of the fixed path mobility model for the MS

```
1  // ...
2  namespace ns3
3  {
4
5  // Mobility model for a node (typically the sink) to follow a predefined path in a cyclic manner
6  class FixedPathdMobilityModel : public MobilityModel
7  {
8  public:
9      // Register this type with the NS-3 TypeId system
10     static TypeId GetTypeId();
```

```
11      // Default constructor
12      FixedPathdMobilityModel();
13      // Set the predefined path for the node to traverse as a sequence of 3D vectors
14      void SetPath(const std::vector<Vector>& path);
15      // Set the number of cycles the node should complete along the path
16      void SetCycleNumber(uint32_t cyclesNumber);
17      // Set the time interval (in milliseconds) between consecutive cycles
18      void SetIntervalBetweenTwoCycles(uint32_t interval);
19      // Get the total tour delay (time taken for one complete cycle) in milliseconds
20      double GetSinkTourDelay();
21
22  private:
23      // Reset the direction and speed of the node for the next segment of the path
24      void ResetDirectionAndSpeed();
25      // Pause the node's movement, cancel any scheduled events, and schedule the end of the pause
26      void BeginPause();
27      // Set the node's velocity and direction for the next path segment and schedule the next pause
28      // \param direction Direction in radians for the node's movement
29      void SetDirectionAndSpeed(double direction);
30      // Initialize the node's direction and speed using random variables
31      void DoInitializePrivate();
32      // Clean up resources when the object is destroyed
33      void DoDispose() override;
34      // Initialize the mobility model before simulation starts
35      void DoInitialize() override;
36      // Get the current position of the node
37      Vector DoGetPosition() const override;
38      // Set the position of the node
39      void DoSetPosition(const Vector& position) override;
40      // Get the current velocity of the node
41      Vector DoGetVelocity() const override;
42      // Assign random variable streams for reproducible random behavior
43      int64_t DoAssignStreams(int64_t) override;
44      // Random variable for controlling the direction of movement
45      Ptr<UniformRandomVariable> m_direction;
46      // Random variable for controlling the speed of the node
47      Ptr<RandomVariableStream> m_speed;
48      // Random variable for controlling pause duration between movements
49      Ptr<RandomVariableStream> m_pause;
50      // Event ID for the next scheduled mobility event
51      EventId m_event;
52      // Helper object for computing velocity and position updates
53      ConstantVelocityHelper m_helper;
54      // Vector storing the predefined path as a sequence of 3D vectors
55      std::vector<Vector> m_path;
56      // Index of the current point in the path
57      size_t m_currentIndex;
58      // Total number of cycles to complete
59      int m_cycle_number;
```

```
60      // Current cycle number being executed
61      int m_current_cycle;
62      // Time interval (in milliseconds) between consecutive cycles
63      int m_interval_between_two_cycles;
64      // Total time taken for one complete tour (cycle) in milliseconds
65      double m_tour_delay;
66      // Time when the current cycle started
67      double m_startingInstant;
68      // Time when the current cycle ended
69      double m_endingInstant;
70  };
71
72  } // namespace ns3
73
74  #endif /* FIXED_PATH_MOBILITY_MODEL_H */
```

A.6 Essential utility functions for simulation

```
1  // ...
2  using namespace ns3;
3  using namespace std;
4
5  // Structure to represent a visiting point for the mobile sink's path in the EDCVP protocol
6  struct VisitingPoint {
7      double x; // X-coordinate of the visiting point
8      double y; // Y-coordinate of the visiting point
9
10     // Constructor to initialize coordinates
11     VisitingPoint(double _x, double _y) : x(_x), y(_y) {}
12
13     // Less-than operator for sorting points (by y, then x)
14     bool operator<(const VisitingPoint& other) const {
15         return y < other.y || (y == other.y && x < other.x);
16     }
17 };
18
19 // Check if the network forms a connected graph based on node positions and communication range
20 bool IsConnectedGraph(NodeContainer& nodes, int communication_range);
21
22 // Generate a random topology for sensor nodes and sink, ensuring minimum distance constraints
23 // Uses NS-3's random number generation without an explicit seed
24 void GenerateTopology(MobilityHelper& mobility, MobilityHelper& sinkMobility, NodeContainer& nodes
       , int L, int communication_range, int min_distance);
25
26 // Generate an SVG image of the grid, showing visiting points and rendezvous cells
27 void GenerateGridImage(const std::string& filename, int k, double spacing, std::vector<
       VisitingPoint>& visitingNodes, std::vector<uint32_t>& rendezvousCells);
```

```
28
29 // Calculate Euclidean distance between two visiting points
30 double distance(const VisitingPoint& p1, const VisitingPoint& p2);
31
32 // Verify if a sequence of visiting points forms a valid Hamiltonian cycle
33 bool isHamiltonianCycle(const std::vector<VisitingPoint>& cycle);
34
35 // Find a Hamiltonian cycle for the given visiting points, if one exists
36 std::vector<VisitingPoint> findHamiltonianCycle(const std::vector<VisitingPoint>& points);
37
38 // Create an adjacency matrix for visiting points based on grid cell size and connectivity
39 std::vector<std::vector<bool>> createAdjacencyMatrix(const std::vector<VisitingPoint>& points, int
       CellSize, int k);
40
41 // Check if adding a vertex to the Hamiltonian path is valid
42 bool isSafe(int v, const vector<vector<bool>>& graph, const vector<int>& path, int pos);
43
44 // Recursively find a Hamiltonian cycle in the graph
45 bool hamCycleUtil(const vector<vector<bool>>& graph, vector<int>& path, int pos);
46
47 // Find a Hamiltonian cycle for visiting points and store it in hamiltonianCycle
48 bool hamCycle(const vector<vector<bool>>& graph, const std::vector<VisitingPoint>& points, std::
      vector<VisitingPoint>& hamiltonianCycle);
49
50 // Compute the total length of a Hamiltonian cycle (sum of distances)
51 double ComputeCycleLength(const std::vector<VisitingPoint>& cycle);
52
53 #endif // HELPER_FUNCTIONS_H
```

A.7 Declaration of helper class for EDCVP routing protocol

```
1  // ...
2  namespace ns3
3  { // Helper class for installing and configuring the EDCVP routing protocol on nodes in NS-3
4  class EdcvpHelper : public Ipv4RoutingHelper
5  {
6  public:
7      // Default constructor
8      EdcvpHelper();
9
10     // Constructor with node-specific parameters
11     EdcvpHelper(std::vector<std::tuple<
12         uint32_t,           // Node ID
13         double,             // X-coordinate
14         double,             // Y-coordinate
15         uint32_t,           // Sink node ID
16         uint32_t,           // Grid cell ID
```

```
        uint32_t,           // Grid cell row
        uint32_t,           // Grid cell column
        uint32_t,           // Maximum communication range
        uint32_t,           // Grid size parameter (k)
        double,             // X-coordinate of cell center
        double,             // Y-coordinate of cell center
        double,             // Initial energy
        double,             // Grid cell size (a)
        double,             // Energy threshold
        uint32_t,           // Time interval to check energy
        bool,               // Flag indicating if node is in a rendezvous cell
        std::map<std::pair<int, int>, std::vector<int>>*, // List of cluster heads for data
    requests
        AnimationInterface* // NetAnim interface for visualization
    >> nodeParams);

    // Create a copy of the helper object
    // Returns a pointer to a dynamically allocated clone
    EdcvpHelper* Copy() const override;

    // Create an instance of the EDCVP routing protocol for a node
    // Called by ns3::InternetStackHelper::Install
    // TODO: Support installing EDCVP on a subset of IP interfaces
    Ptr<Ipv4RoutingProtocol> Create(Ptr<Node> node) const override;

    // Set attributes for the EDCVP routing protocol
    void Set(std::string name, const AttributeValue& value);

    // Assign fixed random variable stream numbers for reproducibility
    // Returns the number of stream indices assigned
    int64_t AssignStreams(NodeContainer c, int64_t stream);

private:
    ObjectFactory m_agentFactory; // Factory to create EDCVP routing objects
    std::vector<std::tuple<
        uint32_t,            // Node ID
        double,             // X-coordinate
        double,             // Y-coordinate
        uint32_t,           // Sink node ID
        uint32_t,           // Grid cell ID
        uint32_t,           // Grid cell row
        uint32_t,           // Grid cell column
        uint32_t,           // Maximum communication range
        uint32_t,           // Grid size parameter (k)
        double,             // X-coordinate of cell center
        double,             // Y-coordinate of cell center
        double,             // Initial energy
        double,             // Grid cell size (a)
        double,             // Energy threshold
```

```
65          uint32_t,           // Time interval to check energy
66          bool,               // Flag indicating if node is in a rendezvous cell
67          std::map<std::pair<int, int>, std::vector<int>>*, // List of cluster heads for data
        requests
68          AnimationInterface* // NetAnim interface for visualization
69      >> m_nodeParams; // Node-specific parameters for EDCVP configuration
70  };
71
72  } // namespace ns3
73
74  #endif /* EDCVP_HELPER_H */
```