TU WIEN Informatics

Area Monitoring with Gossip for Spatially-Distributed Internet of Things

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Alexander Gallauner, BSc
Matrikelnummer 01026090

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar
Mitwirkung: Dr. Christos Tsigkanos

Wien, 20. Jänner 2022

Alexander Gallauner   Schahram Dustdar

Informatics

Area Monitoring with Gossip for Spatially-Distributed Internet of Things

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Alexander Gallauner, BSc
Registration Number 01026090

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar
Assistance: Dr. Christos Tsigkanos

Vienna, 20th January, 2022

Alexander Gallauner

Schahram Dustdar

Erklärung zur Verfassung der Arbeit

Alexander Gallauner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Jänner 2022

Alexander Gallauner

Danksagung

Ich möchte mich besonders bei Christos Tsigkanos bedanken, der mir diese Diplomarbeit initial ermöglicht hat und sich zu jeder Zeit mit großer Geduld mit meinen Problemen und Vorstellungen beschäftigt hat, um konstruktives Feedback geben zu können.

Acknowledgements

I would like to especially thank Christos Tsigkanos, who initially made this diploma thesis possible and dealt with my problems and ideas with great patience to give constructive feedback.

Kurzfassung

Ein üblicher Anwendungsfall im Bereich der Internet der Dinge ist es, Sensoren in einem Gebiet zu verteilen, um bestimmte Phänomene zu beobachten. Diese Art von Anwendung bezeichnen wir als Gebietsüberwachung. Dabei gilt es einige Herausforderungen zu meistern, wie zum Beispiel Energie-Effizienz bei batteriebetriebenen Geräten zu gewährleisten. Außerdem kann es eine Schwierigkeit darstellen, die Sensoren in einer Region so anzuordnen, damit das gesamte Gebiet flächendeckend überwacht wird. In weiterer Folge liegt der Fokus auf dezentralisierte Lösungen, um sowohl Engpässe als auch lange Kommunikationsstrecken zu einer zentralen Stelle zu vermeiden. Weitere Eigenschaften, die durch Dezentralisierung begünstigt sind, sind Stabilität und Ausfallsicherheit des Netzwerkes. Eine Technologie, die diese Anforderungen weitgehend unterstützt und deshalb Hauptaugenmerk dieser Arbeit ist, ist das Gossip Protokoll, das die Kommunikation zwischen den Geräten steuert. Dieses Protokoll bestimmt, dass Daten nur in regelmäßigen Abständen und mit einem zufällig gewählten Kommunikationspartner ausgetauscht werden dürfen.

Bei Implementierungen mit Gossip Protokollen war es in der Vergangenheit nicht unüblich, dass für jedes Problem eine eigene Softwarelösung geschaffen wurde, die oft auf das jeweilige Problem zugeschnitten war und deswegen oft Mängel bei Wartbarkeit und Erweiterbarkeit aufwies. Außerdem ist es oft ein Problem dieser Lösungen, dass diese nicht ausreichend getestet werden und deshalb fehleranfällig sind. Um dem entgegenzuwirken, stellen wir ein Code-Grundgerüst, ein sogenanntes Framework, zur Verfügung, das bereits einige Basisfunktionen in Bezug auf Gebietsüberwachung mit Gossip Protokollen implementiert, jedoch auch auf einfache Weise erweitert werden kann, um komplexeren Szenarien gerecht zu werden. Ein weiterer Zweck des Frameworks ist, dass sich Entwickler nicht mehr mit Details und Prinzipien des Gossip Protokolls beschäftigen müssen, sondern sich voll und ganz den Funktionen des eigentlichen Überwachungsproblems widmen können. Um einen umfassenden Überblick über dieses Thema zu geben und damit zukünftige Entwicklungen unterstützen zu können, identifizieren wir die Hauptanwendungsfälle der Gebietsüberwachung und beschreiben die wichtigsten Charakteristiken. Darauf aufbauend wird eine Methodik vorgestellt, um neue Probleme der Gebietsüberwachung kategorisieren zu können und mögliche unbekannte Faktoren zu identifizieren. Die Erkenntnisse fließen in Referenzimplementierungen, die mit Hilfe des Frameworks erstellt wurden und die Basisfunktionen in Anwendung zeigen. Abschließend werden Simulationen durchgeführt, die aufzeigen, wie sich Gossip Protokolle in Überwachungsszenarien verhalten und welche Parameter welchen Einfluss auf die Gesamtleistung des Systems haben.

Abstract

Within spatially-distributed Internet of Things, a common setting entails devices equipped with sensors being deployed over a region to monitor a specific phenomenon, a problem known as area monitoring. Challenges encountered revolve around optimizing resource usage and achieving full coverage of a target area, while centralized communication is undesirable or infeasible because of operation over wide areas, absence of network coverage or utilization of range-constrained devices. The overall goal is to provide a decentralized infrastructure that is not performance-intensive but robust, predictable and fault-tolerant. To this end, gossip protocols are a solution that fits the problem particularly well. Within a gossip protocol, nodes interact periodically with random peers and exchange information about their respective states.

Usually, developers encode gossip protocol solutions by hand, a practice which is error-prone, leads to non-reusable solutions and an absence of high-level constructs to design, compose and validate implementations – something particularly relevant in the development of spatial monitoring solutions for IoT. In this thesis, we first identify common problems in spatial monitoring, providing methodological support to designers. Subsequently, we provide corresponding reference implementations utilizing gossip. Those are intended as model solutions, and are given in terms of an existing high-level gossip framework which abstracts away low-level details, augmented with various facilities to aid development. Finally, we adopt simulations to demonstrate the effects various parameters of the framework have within spatial monitoring for IoT.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Motivation Scenario 1
1.2 Gossip for Area Monitoring 2
1.3 Methodology & Contributions 3
1.4 Thesis Structure 5

2 Fundamentals and Related Work **7**
2.1 Area Monitoring in Distributed Systems 7
2.2 Decentralized Spatial Computing 10
2.3 Gossip Protocols 13
2.4 MiCA 15

3 Problems in Area Monitoring **21**
3.1 Target Counting 22
3.2 Simple Target Tracking 27
3.3 Continuous Target Tracking 31
3.4 Decision tree for identification 34
3.5 Applicability of the Sensing Models 35

4 Technical Framework for Area Monitoring **41**
4.1 Event control 42
4.2 Network Representation 43
4.3 Logging 44
4.4 Visualization 44
4.5 Packaging 45
4.6 Reference Implementations 46

5 Evaluation **53**

5.1 Reflection on the research questions 59

6 Conclusion & Future Work **61**

List of Figures **63**

List of Tables **65**

Listings **67**

List of Algorithms **68**

Bibliography **69**

CHAPTER 1

Introduction

Nowadays, a relatively common problem in the world of the Internet of Things is area monitoring, where a group of devices equipped with sensors is deployed over a region to gather sensor data about one or multiple phenomena and estimate their current and future positions in the monitored area [NLF07]. The Internet of Things (IoT) [AIM10] is a collective term for a variety of things, like sensors, mobile phones and Radio-Frequency Identification (RFID) tags, which interact with each other to reach common goals. Low resources characterize these smart objects in terms of both computation and energy capacity. Therefore, proposed solutions with smart objects have to pay special attention to resource efficiency and obvious scalability problems. For that reason, a centralized way of communication is not desired because if all nodes communicated with a central node, the communication load, especially over long distances, would quickly drain the network resources and can overwhelm the central node. Consequently, the goal is to provide a decentralized solution that is not performance-intensive but robust and predictable and avoids a single point of failure. Furthermore, there is no demand to provide to-the-second accurate real-time information of the area's state, so new events do not have to be disseminated immediately. Instead, it is enough if the system disseminates them within a few seconds.

Based on these requirements, gossip protocols fit the problem particularly well. Characteristics of gossip protocols are that the nodes interact periodically with arbitrary peers and exchange their respective knowledge. Even though the underlying protocol is straightforward, it offers strong robustness and convergence guarantees. That means a system with a sufficient dense topology converges to a consistent state after logarithmic time, and inconsistencies are limited to recent events.

1.1 Motivation Scenario

To further emphasize the benefits of our work, we would like to address a possible application. Wildlife animal tracking can be a pretty challenging task. However, it is indispensable to learn more about wildlife animals and safeguard their population, especially protected species. In many cases, wildlife animals live in hard-to-reach and harsh regions far from any civilization, where there is no network infrastructure to build on. In addition to the difficulty of observing them, the presence of humans can influence their natural behaviour.

The idea to use IoT in combination with gossip protocols to monitor wildlife animals without disturbing them sounds promising, but some things have to be considered. The cooperation between the nodes in a decentralized way is quite challenging and has to vary from one application to another. Furthermore, the engineer has to provide the functionality to calculate the current position and estimate the target's future position. Additionally, an emergency plan must exist for a possible target loss during tracking. On top of that, the implementation should be energy efficient and scalable. Due to the many requirements, it can be genuinely challenging to realize a wildlife animal tracking application. The engineer has to be familiar with the details of the field to provide a reasonable and accurate solution. Also, most existing gossip solutions are designed for a particular problem and software engineering principles like maintainability and extendibility were not considered. That makes it more arduous to take advantage of existing solutions and reuse parts of other implementations.

1.2 Gossip for Area Monitoring

The main idea of this paper is to assist the implementation of new applications for area monitoring with gossip in a network of spatially-distributed IoT. We have decided to use the gossip protocol for communication because it shows some characteristics that are beneficial for decentralized infrastructures. To list a handful of strengths, elaborated by Birman [Bir07]:

- It is simple.
- It generates a bounded load on participants.
- It is independent of the underlying topology.
- It is robust to transient network disruptions.
- It is convergent consistent.

Of course, gossip also has its downsides like its relatively slow exchange rate, but it is the purpose of this thesis to show the competitiveness of this protocol in area monitoring applications. In order to provide support for the realization of new applications, a critical outcome should be a methodology, which can identify nearly any problem in the field of area monitoring of spatially-distributed smart objects. Generally, we derived following use cases from a great variety of scientifc papers about area monitoring:

- Target counting

- Simple target tracking
- Continuous target tracking

These use cases are tackled most often in literature and should cover most likely all problems that can arise in area monitoring. Dependent on the known requirements of a new application, the methodology suggests a design pattern. A design pattern reflects both the essential key points of a use case and unique demands that an engineer has to consider. Therefore, a use case can have one or more design patterns. For each design pattern, we provide eligible and well-known area monitoring algorithms.

To make the realization of an application even more straightforward, we provide a technical framework that can be used as a basis for new implementations, because it provides features that all of our analyzed use cases have in common. This framework builds on a Microprotocol Composition Architecture (MiCA), a system of Princehouse et al. [PCJ$^+$14] for building gossip-based tools that are highly resistant to disruptions and make efficient use of system resources. MiCA realizes abstractions regarding communication and simulation on top of the Java Virtual Machine, so developers do not have to worry about low-level details. In addition to that, MiCA implements optimizations that significantly reduce the number and size of messages used.

To get familiar with the technical framework and to show usability and effectiveness, we provide sound reference implementations. The reference implementations should help to understand the principles of the framework and what steps a developer must take to provide a solution for a specific problem. Another feature of the framework is that it also enables advanced simulations of the implementations to test the performance and correctness of the solution. It offers different ways to fill the system with sample sensing data and enables special logging features to ease possible debugging work and to retrace the impacts of sample input data.

1.3 Methodology & Contributions

The expected results of this thesis consist of three parts:

- Our work provides a methodology for the systematic identification of area monitoring problems. The application of this methodology results in a suggestion of a design pattern. Each design pattern is well documented, the characteristics are known, and one or more suitable area monitoring algorithms are given.
- We provide a technical framework to support the implementation of new gossip-based area monitoring applications.
- We evaluate how different network structures influence the dissemination behaviour of gossip for area monitoring. Is there any network configuration that best fits gossip protocols?

We frame the expected results into the following research questions, which are covered in the entire thesis, but definitely answered in Chapter 5:

RQ1. What are the main problem and application areas of area monitoring within the Internet of Things?

RQ2. How can we support engineers to develop area monitoring applications with gossip for the Internet of Things?

RQ3. What is the effect of different network structures for area monitoring with gossip protocols?

We use the following methodological approach to achieve these objectives:

- First of all, we deal with the fundamentals of area monitoring. That should help to subdivide area monitoring into prominent use cases. If we found an additional primary use case during research, we would add that use case to our list of well-known use cases.

- We analyze a variety of existing area monitoring problems and applications and derive requirements. Afterwards, we categorize these requirements.

- Building on the work done so far, we create design patterns. A design pattern belongs to precisely one use case and outlines the main characteristics of that use case and specific requirements. For this reason, there can exist more than one design pattern for a use case. Design patterns of the same use case are very similar but differ in one or more critical points that affect the method of area monitoring.

- The information collected so far helps to provide a methodology for the identification of area monitoring problems. Based on the known requirements of a "new" application, the methodology picks a design pattern to promote the understanding of the critical points and traps of the application.

- To ease the implementation of area monitoring applications, we provide a technical framework. At this point, we are aware of the main components of the system and can plan the structure of the framework in a meaningful way.

- After building up the framework, we provide expressive reference implementations for distinct area monitoring problems. Issues arising during development are documented. These issues will influence the improvements of the framework. We apply the testing and simulation feature of the framework to demonstrate the functionality of the application.

- In the end, we evaluate the usability of our methodology and technical framework and grade the applicability of our contributions (including the use of gossip) to a variety of different network structures.

1.4 Thesis Structure

After giving a brief overview in this chapter, we will go deeper in Chapter 2 and explain the most relevant objectives a distributed area monitoring solution should fulfil to work correctly. Also, we give a preview of the observed area monitoring patterns, from which we think they are the most relevant ones. We also present metrics to compare different area monitoring algorithms. Additionally, Chapter 2 explains the basics of decentralization and the difference to a distributed system. As gossip protocols play an essential role in this thesis, we dedicate an own section to show the main features, strengths and limitations of gossip. On top of that, we present MiCA to ease the implementation of gossip applications and to show the benefits of such a high-level framework.

Chapter 3 presents a methodology to identify area monitoring problems dependent on their requirements. That is done by explaining relevant aspects of our elaborated use cases to identify an area monitoring problem. With the help of our developed decision trees, it is possible to handle specific requirements and provide existing algorithms. Furthermore, we compare the different sensing models to find the most appropriate model for a given problem.

In Chapter 4, we present our technical framework that eases the implementation of new area monitoring applications with gossip. We will review the most important features that make this framework unique and show three example implementations that were built with it and contributed to its characteristics.

Chapter 5 gives a deeper insight into how gossip protocols perform with different network structures. Therefore, we compare three different network structures and show how possible network restrictions influence the behaviour and response time. In addition to that, this chapter revisits the research questions of this thesis and answers them.

In the end, Chapter 6 will give a conclusion of the thesis and present plans for future work.

CHAPTER 2

Fundamentals and Related Work

This chapter will cover the fundamentals of area monitoring and the most common features. Further, it shows the principles of decentralization and how gossip protocols can provide decentralized systems. Finally, we will introduce MiCA to support the implementation of gossip-based systems.

2.1 Area Monitoring in Distributed Systems

In this work, we deal with the main tasks and challenges of area monitoring with spatially-distributed IoT. These smart objects usually are resource-constrained devices, which have sensor capabilities. The built-in sensors can monitor their environment in different goal-oriented ways to collect, process and exchange data. In contrast to a more sophisticated surveillance technology like RADAR (Radio Detection and Ranging), which is robust, reliable, accurate, but expensive, IoT enables cheaper solutions that do not rely on any centralized infrastructure.

Area monitoring has different applications, such as intruder tracking, surveillance, traffic monitoring and habitat monitoring. This enumeration of possible applications could still be continued by a large number of examples, but in fact, we can reduce all applications to three prominent cases:

- **Target Counting:** This use case's main task is to count objects of a specific type in a predefined area.
- **Simple Target Tracking:** Often builds upon target counting. Many target tracking algorithms need target counting to work correctly, but the mission should be to track objects continually as long as they are in the monitored area.
- **Continuous Target Tracking:** Within continuous target tracking, the target type is not limited to simple objects, such as vehicles or animals. In some cases, it is necessary to track phenomena like forest fires.

Chapter 3 will dedicate a section to each of these cases to point out the requirements, main characteristics, the challenges and current applications and algorithms.

2.1.1 Parts of an area monitoring system

Many components influence the way of communicating and predicting targets. Souza, Nakamura and Pazzi [SNP16] explain six features that are fundamental for area monitoring algorithms:

- Target Detection
- Node Cooperation
- Current-Position Estimation / Target Localization
- Future-Position Prediction
- Energy Management
- Target Recovery

Further explanations of the features can be found in the sections of the primary use cases in 3.

The development of a new area monitoring application should always begin with identifying the correct use case. That makes it easier to plan and specify requirements for such a system. Based on the conditions, it is possible to decide which components are needed to satisfy the system's needs. On top of that, it is possible to limit the selection of possible algorithms to fulfil the task. Not every application requires the implementation of all features. In any case, Target Detection and Node Cooperation belong to the essential components that every system should have. The other components can improve an application's overall performance to provide a more long-lasting and accurate area monitoring system but are not vital for an area monitoring system's basic functionality.

2.1.2 Secondary Objectives

This section will introduce some objectives, which are often assumed to be ideally satisfied in many existing algorithms. Problems regarding these objectives can cause severe problems to the final result of the application. Souza, Nakamura and Pazzi [SNP16] summarized the following secondary objectives:

- **Density Control:** Goal of the density control is to find the minimum set of nodes that have to be active to maintain the network functionality. The activation of all sensor nodes would lead to high energy consumption and highly correlated and redundant messages. So to say, there is a trade-off between energy usage and coverage. Shang and Shi [SS05] considered this trade-off and presented algorithms that maximize coverage while minimizing the overlap between the sensor regions. Campos et al. evaluated in their work [CSN$^+$12] how density control and localization system affects an application's behaviour. But it can also be seen in 3 that the used sensing model influences the impact of sensor overlaps.

- **Clock Synchronization:** Job of the clock synchronization is to achieve a consistent timescale for local clocks of all nodes in the network. Due to physical variations in the sensors' environment, the computer clock drifts and the local clocks of the different nodes may become inconsistent. Because some algorithms need a consistent clock for all nodes, we need to counteract that with clock synchronization.

- **Localization System:** The localization system is responsible for providing sensor nodes with position information. In target tracking, this information has a decisive role because it is needed to compute the target's current and future position. Often, just a couple of nodes know their location a priori, because they are equipped with GPS or consciously placed in a specific position. These nodes are also called beacon nodes. Localization algorithms use the position of the beacon nodes to estimate the location of the other non-beacon nodes.

- **Security:** The common security goals are confidentiality, integrity and availability for all messages. The information that is communicated in the sensor network should not be available to unauthorized individuals. Furthermore, it should not be possible to modify the information in an unauthorized manner. Also, the information should be available when it is needed. However, security features in decentralized area monitoring are expensive, like Du and Chen [DC08] explain in their work. For some application types, the security system plays a major role, like military and security-monitoring applications.

2.1.3 Performance & Efficiency Metrics

In this section, we explain the fundamental metrics to analyze and compare the performance of area monitoring algorithms. Of course, the choice of the metrics depends on the goal of the application. Souza, Nakamura and Pazzi [SNP16] mentioned the following metrics:

- **Delay:** Beside precision, this metric belongs to the primary metrics because it is crucial for the area monitoring outcome, especially for real-time tracking applications. Roughly described, the delay states out how long the system takes to recognize and compute a target.

- **Precision:** Precision measures the difference between the estimated and actual target position, or in case of target counting estimated and actual target count. For target tracking applications, this metric is also applied to the predicted position of the target. At a particular time in the future, the expected position is compared to the target's actual position. A higher prediction error can influence the operability of the energy management component. In many algorithms, the prediction results are used to deactivate superfluous nodes to save energy. Inaccurate prediction results will activate a wrong set of nodes and possibly cause a loss of the target.

- **Communication Cost:** Communication costs result from the number of necessary messages to build up the network and track or count targets. Because communication is quite energy-consuming, most algorithms try to reduce communication in the system. Different factors can influence these costs:
  - *Network structure:* Herefore, the network is structured in a way that enables data fusion.
  - *Data precision and data reporting frequency:* It is a fact that a higher data precision and a higher data reporting frequency results in higher communication overhead. A trade-off has to be done.
  - *Number of targets:* Also, the number of targets influence the communication overhead. Data fusion is here a vital countermeasure.
- **Energy Consumption:** Energy consumption is an important topic in sensor networks and should not be neglected. As already mentioned in the description of the metric communication costs, communication makes up a large part of an area monitoring system's energy consumption. Data fusion is a crucial method to reduce communication. Moreover, the prediction of target positions and the related deactivation of unnecessary nodes is quite useful. A lost of a target should be avoided because target recovery can be truly resource-intensive.

2.2 Decentralized Spatial Computing

The term *spatial computing* has long been used to express computing with information about (geographic) space, as explained by Duckham [Duc13]. Therefore, conventional approaches to spatial computing are founded on the assumption that spatial information is stored and processed in large information repositories, such as GIS (geographic information system) or spatial databases. Thus, the location of the repository is irrelevant to the information that is stored in that repository.

The term *spatial computing* recently evolved and became the term for computing with information in and about (geographic) space. This description introduces the *computing somewhere* paradigm, where information about geographic space is also distributed throughout this space. In addition to that, it presents spatial information neighbourhoods, where information exchange is preferred between spatially closer locations and therefore constraints data movement. It is easy to see that this idea rejects the traditional way of spatial computing in centralized spatial information repositories.

To take a step further, we introduce *decentralized spatial computing*, which puts an additional constraint on spatial computing. Therefore it is essential to distinguish between a distributed and a decentralized system. A distributed system declares that the components of the system cooperate to reach a common goal [WD04]. Decentralized systems are a subset of distributed systems where no single member knows the entire system state [Lyn96]. Therefore, a decentralized system meets the requirements of *computing somewhere* very well regarding the constraint on data moving. For example, it would be undesirable to collect the complete system state in a single point because that would mean that it is necessary to move information over long distances. Decentralized systems meet this requirement so that instead of communicating the data to a central unit, the components process and filter data locally accessible to them, which entails a high degree of autonomy. However, it is essential to know that there are different degrees of decentralization. A fully decentralized system would completely isolate the components of the system, so no information is shared between the individual members. In comparison to that, a fully centralized system would imply that a single member knows everything. That one of these extremes is used by an application is pretty rare. There are more highly but not fully decentralized or centralized systems in real-world applications.

Besides the fact that processing data needs less energy than communicating it, Duckham [Duc13] lists five other reasons for choosing decentralized spatial computing:

- **Information overhead:** It can be the case that there are thousands of nodes in the network, which produce information about the environment. That can lead to a situation where the system is overloaded with information and cannot process the data in time. Decentralization counteracts this situation by filtering and aggregating the data before communicating it.
- **Scalability:** To increase the level of spatial detail in an application, it is necessary to provide a higher number of nodes, which capture information about their environment. Central coordination can come to its limits with the increasing number of participating nodes. In addition to that, a requirement for deploying thousands of nodes is that the nodes are cheap. Such nodes are highly prone to failure, so the system has to adapt automatically to node failures. In the case of centralized control, it is harder to react to such changes in the system. Decentralization offers a solution to these problems as it enables the system behaviour to be defined through interactions between individual nodes. So it can respond better to changes in the system than a central coordinator would do, which has to record and coordinate every change.
- **Sensor-actuator networks:** A sensor/actuator network does not just capture data. It also changes the state of the environment. An example can be a system that senses the soil moisture of a huge area. If some parts of this area are too dry, the system can react and provide water. A centralized system would waste energy to communicate the information to a central node to process the data and send the processed data and actions back to the actuators. With a decentralized solution, the data would be processed and actions would be set close to the location where the information is generated.
- **Latency:** Collecting and processing information in a centralized manner takes time. The slowest message controls the overall time of the system to process information and react to new events. This time can be defined as the system or operational latency. In real-time applications like mobility information services

that calculate public transport routes, it can be critical to react quickly to events. Therefore, decentralized solutions can reduce the operational latency by shortening the information's path and reducing the amount of communicated data.

- **Information privacy:** Decentralized spatial systems can provide information privacy without communicating personal information to a central node. Instead, the data is processed locally or just shared with nearby nodes.

A big motivation for the *computing somewhere* paradigm were geosensor networks. Nittel et al. [NSC$^+$04] describe a geosensor network as a wireless sensor network that monitors phenomena in geographic space. A wireless sensor network is defined as a wireless network of miniaturized sensor-enabled computers, also called sensor nodes [ZGG04].

A sensor node must consist of the following parts:

- a microcontroller for processing data
- a wireless radio for communication data
- at least one sensor for capturing data about its environment

A sensor is a device that measures the physical qualities of the environment and converts environmental stimuli into a digital signal. We can distinguish between the following types of sensors, which generally go along with the environmental stimulus they capture: Acoustic, Chemical, Electromagnetic, Optical, Thermal, Mechanical. We will distinguish throughout this paper between the term *sensor node* and *sensor*. We use the term *sensor node* for a device that is capable of the *three geographic Cs* (capture, communicate, compute) and the term *sensor* for a device that is capable of capturing an environmental stimulus.

Duckham [Duc13] describes several characteristics of geosensor networks, making them unique and advantageous to traditional approaches.

- Flexibility is a crucial factor when deciding on geosensor networks. They are adaptable to a broad range of application scenarios because they are low-cost and can be nearly deployed anywhere.
- Geosensor networks are automated. They can be deployed and used anywhere, where it is too costly or dangerous to involve human interaction. The sensor nodes have an onboard power source and automatically communicate wirelessly to achieve a common goal.
- Geosensor networks can capture information of great spatial and temporal detail because they are close to the phenomena they monitor. As a result of the flexibility of geosensor networks, they can also adapt their spatial and temporal granularity when something interesting happens in their environment.

In contrast, some problems arise when using geosensor networks:

- Normally sensor nodes in geosensor networks are untethered. So their lifespan relies on the battery capacity of the node and the energy consumption of the system operations. So a strategy in many geosensor applications is to put idling nodes into sleep mode and wake them up when needed.

- A significant constraint in wireless sensor networks is that wireless communication is the most energy-intensive operation that a sensor node can perform. Communicating a single bit needs about 1.000 times more energy than the energy required to execute a single microcontroller instruction, like Hill et al. [HSW$^+$00] describe in their work. In addition to that, the average energy lost by a wireless signal is on earth d^4, where d is the distance between two communicating nodes, which means that nodes should transmit only over short distances in geographic space. A workaround for that can be multi-hop communication.

Finally, the combination of geosensor networks and decentralization gives an ample opportunity for decentralized spatial computing and the related *computing somewhere* paradigm. Furthermore, the limited battery life of portable devices and the error-proneness of sensor nodes make decentralized approaches beneficial compared to centralized ones. Like already mentioned, geosensor networks have many different use cases. Area monitoring is one of them, and geosensor networks play an essential role in this application. In the next section, we will dive deeper into the principles of the gossip protocol and how we can use it to establish a decentralized solution in a geosensor network.

2.3 Gossip Protocols

This section aims to show the central aspects and rules of gossip protocols, extensively described by Birman [Bir07]. Firstly the nodes in a gossip-based system must interact periodically with a peer to exchange their respective state. The interactions between nodes are always pairwise, and the state changes should always reflect the state of the other one. If one node pings another one, it does not fall under a gossip interaction. The frequency of how often nodes communicate with each other is pretty low compared to the usual message latency. Because of that, the costs of running such a protocol are negligible. In a typical gossip protocol, the nodes communicate about a specific event for r rounds, which can be defined as the fanout of the protocol. In many gossip-based systems, the fanout is $log(N)$ in a system with N nodes.

Furthermore, it is a significant characteristic of gossip protocols that the nodes select their peers randomly [Bir07]. That does not have to be a uniform random selection of the global set of nodes or the set of neighbouring nodes. The probability distribution can be tailored to particular needs in the system and can be used to build up overlay trees on top of the physical network. In general, there are three application types for gossip protocols, dissemination protocols, anti-entropy protocols and aggregation protocols.

- *Dissemination protocols* are the most popular ones and also the easiest to understand, described by Heinzelman, Kulik and Balakrishnan [HKB99]. Gossip is used to distribute information and flooding the network. Within dissemination protocols, there is a distinction between two types:

Event dissemination protocols are used to spread out events in the network. An important fact is that the events do not trigger the communication, communication always happens periodically.

Dissemination protocols which communicate background data of the nodes belong to the second type. However, protocol propagation latency is not a relevant factor because the information that is carried out changes slowly.

- Mentioned by Birman [Bir07] a type of gossip protocols are *anti-entropy protocols* to resolve inconsistencies in the system, by checking the differences between the nodes.

- *Aggregation protocols* explained in more detail by Kempe, Dobra and Gehrke [KDG03], pick samples of the nodes in the system and combine or reduce them to global value. For example, that can be the minimum value of a specific property or the highest CPU load of all nodes. The aggregation result can also be much more complicated. A widespread use case is that gossip is applied to construct an overlay tree on top of the physical network to provide a logical network that fits the needs for a specific application.

  Besides these different applications of gossip protocols, we must have in mind that gossip protocols run in a periodic, relatively lazy manner where the peers are selected randomly. Another significant fact is that gossip protocols with exponentially rapid convergence towards a state that arises with probability 1.0 are the most useful ones.

2.3.1 Strengths of Gossip

Gossip protocols can be beneficial because of different strengths they come up with, elaborated by Birman [Bir07].

- **Simplicity:** A significant advantage is that the principles of gossip protocols are straightforward and therefore gossip protocols are simple to implement.

- **Robustness:** Although gossip-based systems are not robust to all kinds of failures, transient network disruptions and a small number of lost messages should have little to no effect on overall convergence, because there are exponentially many routes by which the information can flow from the source to its destination. On the other side, dissemination protocols can be vulnerable to data corruption and aggregation protocols to computational errors and faulty nodes. To minimise or even eliminate such problems, it would be necessary to have a backup protocol besides the primary gossip protocol.

- **Convergently consistency:** As already mentioned, properly designed gossip protocols have an exponentially rapid convergence towards a state that emerges with a probability of 1.0. If they are not overwhelmed by a higher rate of incoming events than the system bandwidth can carry, they have a logarithmic mixing time of $O(log(N))$, where N is the size of the system. It is the task of the designer of the system to consider the bounded gossip rate and message size to build up a system which is not overwhelmed by a flood of incoming events.

- **Topology independence:** Gossip protocols can run on a vast number of topologies if these topologies are adequately connected and have sufficient bandwidth.

- **Bounded load:** The problem of many protocols is that the load on individual nodes is pretty high. The strength of gossip-based systems is that the load on nodes is bounded and usually is just a fraction of the overall bandwidth. Here the main characteristics play an essential role. Gossip comes up with a bounded message size and a lazy periodic and random message exchange between nodes. In addition to that, the work between the nodes is evenly distributed.

2.3.2 Limitations of Gossip

Besides the strengths of gossip protocols, there are some limitations of gossip that Birman [Bir07] introduces. For example, it is obvious to see that the small, bounded message size and the relatively low message exchange rate can lead to restrictions in the communication in a gossip-based system.

Generally, we can say that a gossip multicast keeps the N nodes for $O(log(N))$ rounds busy. So the average rate at which new events can be initiated will be estimated the inverse of the gossip multicast time $1/log(N)$. Furthermore, if the information that a node has to gossip exceeds the bounded message size, the effective fanout will start to drop, because the nodes are not able to gossip the information in the expected number of rounds.

Another already mentioned problem is the relatively slow rate of gossip protocols. It would be easy to tune the rate to fit the needs for the application, but that can lead to malfunctions in the system, the overhead for running the protocol would increase dramatically. Also, this would break the rule for gossip that the frequency of how often nodes communicate with each other is pretty low compared to the usual message latency.

2.4 MiCA

The principles of gossip protocols are pretty simple. Nevertheless, it can be quite challenging to develop an efficient and error-free solution that fits all needs. For this reason, we decided to use an existing framework that is already field-tested. We take a more in-depth look at MiCA (Microprotocol Composition Architecture), introduced by Princehouse et al. [PCJ$^+$14]. MiCA is a system for building gossip-based systems that are highly resistant to disruptions and make efficient use of the system resources. For that reason, these systems are very suitable to deal with decentralized tasks.

The main difference to custom-made gossip protocols is that MiCA offers abstractions to express the main characteristics of gossip protocols so that developers can concentrate on the high-level details of their application. Additionally, it offers many composition operators to merge protocols to a more sophisticated protocol. These abstractions are realised on top of the Java Virtual Machine and introduce optimizations regarding the communication in composited protocols. Generally, MiCA describes gossip protocols in terms of three methods:

- **View:** This method controls peer selection during an exchange. By default, it is a uniform random selection from the set of neighbours of a node, but it also allows to specify a discrete probability distribution on the set of network addresses. To select the peers by a discrete probability distribution produces more flexibility and can be used for instance to encode notions of locality, so to say to express that the nodes should communicate more frequently with nearby neighbours. Furthermore, it can be used to encode overlay topologies on top of the physical network.

- **Update:** The update method takes the state of the peer as input and performs an exchange. It is vital to know that this operation modifies the initiating node and the peer. The exchange is not an atomic operation. Due to failures, only one node may be updated.

- **Rate:** The rating method gives information about how frequently exchanges should occur. More precisely it specifies the gossip rate of a node relative to the basic unit of time. The default rate in MiCA is 1.0.

2.4.1 Example Implementation

To get a better insight into MiCA, we show an example of a MiCA implementation in Listing 2.1. In this example, a protocol is implemented that shares both the coordinates and the neighbours of the sensor node. For the sake of simplicity, each node has a static position and gets the position during initialization. However, this protocol would also provide correct information if the sensor node is mobile and moves in the sensing region.

The field `nodeId` identifies a node. The map `aggregate` contains all the information the node collected so far by gossiping. This field will be modified during an update with `void update(Protocol that)` and puts the localization information of a gossip partner into the map. The map uses the address of a node as a key for the key-value pair.

It is important to know that the whole state of the gossip partners is synchronized. So we do not just get the position and neighbours of the gossip partner. We also get the information of the sensor nodes the current gossip partner communicated so far.

Usually, we save the overlay information and, therefore, the neighbours of a node in the `view` field. That information is accessible with the `getView()` method inherited from the `BaseProtocol` class.

```
public class Localization extends BaseProtocol {
    @View
    public Overlay view;
    private final int nodeId;
    @Getter
    private final Point currentPosition;
    public Map<String, LocalizationInformation> aggregate = new HashMap<>();


    public Localization(Overlay overlay, Point point, int nodeId) {
        super();
        this.view = overlay;
        this.currentPosition = point;
        this.nodeId = nodeId;
    }

    @Override
    public void update(Protocol that) {
        Localization other = (Localization) that;

        exchange(other);
        other.exchange(this);
    }

    private void exchange(Localization other) {
        exchangeOldEvents(other);
        exchangeNewEvents(other);
    }

    private void exchangeOldEvents(Localization other) {
        this.aggregate.putAll(other.aggregate);
    }

    private void exchangeNewEvents(Localization other) {
        var localizationInfo = LocalizationInformation.builder()
            .point(currentPosition)
            .nodeId(nodeId)
            .neighbours(
            view.getView().keySet().stream()
                .map(String::valueOf)
                .collect(Collectors.toSet())
            )
            .build();

        other.aggregate.put(getAddress().toString(), localizationInfo);
        this.aggregate.put(getAddress().toString(), localizationInfo);
    }
}
```

Listing 2.1: Example implementation of a protocol with the high-level framework MiCA. The node that runs the protocol instance shares its current position and network related information like its current neighbours.

The significant advantage of MiCA is that we know that the main characteristics of gossiping are satisfied and we do not have to worry about low-level details. It is sufficient to implement the update method to realize a simple gossip protocol.

2.4.2 Composition operators

Besides the simple implementation of gossip protocols, MiCA offers a collection of composition operators that combine several smaller protocols into a larger one. Without a high-level framework, it would be necessary to implement the composition by hand, which can be error-prone and also has a lack of optimizations compared to an efficient high-level framework.

Generally, it is possible to combine protocols in different ways. The composition operators can be categorised along two axes, whether the communication and state of the composed protocols are shared or isolated. MiCA takes advantage of both sharing state and communication between the composed protocols. MiCA supports that a state of one protocol is dependent on the state of another protocol. Moreover, the composition operators can make use of bundling messages for communication optimisations.

MiCA offers different operators to merge small protocols to a larger sophisticated protocol. The most relevant ones, regarding the defined problem of this work, are following (specified by Princehouse, Lonnie, et al. [PCJ$^+$14]):

- **Round-Robin Merging:** Protocols are merged in that way that their operations are interleaved in round-robin fashion. So if we had two sub-protocols, the round-robin composition operation would alternate between the exchanges of the two. This composition operator has a restriction. It can just merge protocols with the same rate. Otherwise, it would be incorrect to combine them in round-robin fashion.

- **Correlated Merging:** A merge operator who is not dependent on the rate of the sub-protocols, is the correlated composition operator. The way to combine the protocols into one is to do so in a probabilistic way. It takes the view and rate method as input and calculates a composite distribution that represents the peer selection of both. This composition takes the advantage that sub-protocols may gossip with the same peer, allowing that the messages are bundled to a single exchange and reducing the overall communication overhead. Like explained in [PCJ$^+$14], this composition operator is very aggressive in trying to exploit this form of overlap. Correlated merge has two relevant advantages. It does not make any assumptions about the protocols being combined, and it can reduce the communication overhead with the bundling of messages. Nevertheless, there is also a downside to this style of composition. Because of the aggressive way to overlap the messages of the sub-protocols, the peer selection preferences of the sub-protocols are no longer independent. The resulting dependency can be a problem for some applications, where the sub-protocols need this independence. To give an example, random walk protocols, developed by Massoulie et al. [MLMKG06], need the independence of peer selection to work correctly. We would not want to have that the random walk sub-protocols would generate the same walks.

- **Independent Merging:** To solve this problem, MiCA also offers an independent composition operator. Like correlated merge, it has a probabilistic peer selection and combines the view and rate method of the sub-protocols. It just bundles messages, if the sub-protocols select the same peer in a given round.

2.4.3 Correctness Properties

To ensure that the protocols in a composition work correct, Princehouse et al. [PCJ+14] claims that there are three correctness properties which must be fulfilled by a gossip composition.

- **View preservation:** The ratio of the frequencies of initiating gossip exchanges that update the sub-protocols are identical to the ratio of the distributions of the sub-protocols.
- **Rate preservation:** Ensures that each sub-protocol runs at the same wall-clock rate as it would run in isolation. This property can be easily mistaken with the first-mentioned requirement. To ensure the view preservation property, the composite protocol only executes each sub-protocols on certain exchanges, while to ensure the rate preservation property, it must not delay the rate of each sub-protocol.
- **State preservation:** Ensures that the composition does not cause any mingling of the states of the sub-protocols. However, that does not mean that a protocol can not use the state of another protocol as a read-only input, for instance, when communicating in an overlay topology.

To summarize, MiCA is a helpful framework that provides a well-functioning gossip layer with unique features like the composition of protocols. Therefore, it is not necessary to bother with low-level details. From now on, the focus can be on the main problem of this thesis: Area Monitoring with Gossip for Spatially-Distributed Internet of Things.

CHAPTER 3

Problems in Area Monitoring

A crucial contribution of this master thesis is a methodology to identify area monitoring problems. An essential prerequisite for such a method is to have a comprehensive knowledge of area monitoring to derive the main characteristics, distinguish between the primary area monitoring patterns, and highlight the connections between them.

Similar to the approach of [MTP$^+$19], where Menghi et al. identified specification patterns for robotic missions, we identified area monitoring problems as follows:

- We do qualitative research on existing scientific papers and applications. Therefore, we looked for the most popular survey papers of area monitoring covering the state-of-the-art.
- We studied them to build a knowledge base of area monitoring. The goal was to derive a distinction of the use cases and to highlight the main characteristics. We already gave an overview of the results in the chapter for area monitoring fundamentals 2.1.
- The most important references were treated recursively. That means that we studied the references of the survey papers in the same manner to expand the existing knowledge base. Further, the subsequent references were treated until we could not add any new information to the knowledge base.
- The next step was to identify the area monitoring patterns. A pattern is characterized by:
  - a name
  - the description of the pattern that should clarify the intent of the pattern
  - required and useful components
  - examples of known uses

- On top of that, we provide decision trees to simplify the implementation of an area monitoring solution and filter preexisting algorithms according to the requirements.
- At the end of this chapter, there is a comparison of the different sensing models and suitable conditions.

3.1 Target Counting

Target counting belongs to the most fundamental problems in wireless sensor networks and estimates the total count of targets in a particular monitored region. Counting targets is usually the first step in many other applications like target tracking, as stated out by Gandhi, Kumar and Suri [GKS08]. A target could refer to not only materialized objects but also to specific activities such a signal in transmission or the "power-on" status of a device. The difficulty of target counting lies in obtaining the exact target count number. The main challenge is to cover the area of interest fully, which unavoidably results in overlaps in the sensing region and as a consequence, leads to detection of the target by more than one sensor. Additionally, it is not straightforward to infer the exact target count from sensor measurements.

Within this work, we decided to declare Target Counting as a primary use case besides Target Tracking. Gandhi et al. [GKS08] give two reasons for that:

- Target counting is a major problem in its own right. It is a fundamental end goal in many surveillance applications, accurately estimating a population, for instance, animals in natural habitats or intruders in sensitive areas.
- A reliable estimate of the target count can be the groundwork for robust target tracking. Tracking heuristics based on particle filters need an educated guess on the number of unknown targets to avoid getting stuck.

There are many applications, where a target counting system can meet all requirements and establishing a target tracking system would be a waste of resources. Target counting offers room for many optimizations regarding hardware usage, node computations and network communication. Furthermore, there is an inherent tension between counting and localization accuracy regarding the overlapping ranges of sensor deployments. While in most cases, target counting can be improved by minimizing the overlap among different sensing ranges, target tracking performs better with increasing the overlap [SMS06].

3.1.1 Components

Section 2.1 mentions the most significant components of area monitoring. In the case of target counting, not all of them are crucial for providing a well-functioning system. Within target counting, we will concentrate on Target Detection, Node Cooperation, Target Localization and Energy Management, even if Target Localization is not an absolute requirement like in target tracking systems, but can support other components. Future-Position Predication and Target Recovery can be neglected because it is a mechanism that is closely connected to the principles of target tracking and is usually not a requirement for the current use case.

Target Detection

In the field of target counting, there are various types of sensing models. Wu, Zhang and Cheng [WZLC14] classified existing works into four major categories based on the estimation methods and the sensing capabilities.

- **Binary counting:** The binary counting model relies on binary sensors, which can detect the existence of one or more targets within its sensing range and outputs "0" (absence of any target) or "1" (presence of one or more targets). Because of the 1-bit state representation of a sensor, the communication overhead is very low compared to other models.
- **Numeric counting:** Within this model, sensor readings can count the total number of targets in their sensing range. This functionality is typically provided by photo-electric technology.
- **Energy counting:** Like compressive counting, this counting model measures the signal strength of the target energies at each sensor to estimate the target count. It uses traditional machine learning or statistical techniques to estimate the count.
- **Compressive counting:** Like energy counting, it measures the target's energy, but on the contrary, it uses, with the exploitation of the Compressive Sampling (CS) theory, a different estimation method. Like described by Cands and Wakin [CW08], CS can reconstruct a sparse signal from a small number of measurements within polynomial time.

Before selecting a target counting algorithm for a new real-world application, it should be noted that every counting model is just suitable for some operating environments. For example, numeric counting is just applicable for targets with visible contours, which numeric photo-electric sensors can detect. On the contrary, energy sensors should be considered if the targets can not be detected visually, like radioactive devices.

To make things more understandable and better predictable, most algorithms assume a so-called idealized sensing model. This idealized sensing model addresses two of the most critical assumptions:

- Unit disk sensing ranges
- Perfect target count

Of course, there are also algorithms that can handle uncertainties, like the algorithms from Gandhi et al. [GKS08]. They considered sensing ranges that are neither symmetric about the centre nor unit disk. Additionally, they can handle noisy target counts by sensors.

Node Cooperation

This component aims to reduce the number of messages that the nodes of the network communicate. Fewer messages lead to lower communication overhead and less energy consumption. Data aggregation is the principal concept to reduce messages in a target tracking system. Beside a centralized approach, where all sensor nodes send their readings towards the sink nodes, and an increasing number of sensor nodes can overwhelm the sink nodes, the logical structure of sensor networks is usually organized as a cluster, tree or faces.

- Cluster-based [XL07] [MISR11]: Clustering enables data fusion, which implies reducing the communication overhead. As a consequence, it supports high scalability of nodes. A cluster consists of a cluster head and cluster members. The cluster head is responsible for collecting the members' information and fusing it to a single data report. Afterwards, the cluster head sends the report to the specified sink node of the network. In theory, there are three types of clustering: static, dynamic and hybrid.
  - In a static cluster, clusters are organized at the time of the target tracking system's deployment. Size and members of the cluster do not change during the lifetime of the system. Other things like the cluster head may vary for the benefit of energy efficiency. Although this type is straightforward and reduces the communication overhead, it has a low tolerance to failures. Also, nodes of different clusters can not communicate with each other, which leads to redundant communication when an event occurs in more than one cluster at the same time.
  - Dynamic clusters adapt themself dependent on the target that is moving in the monitored area. In this approach, the members of a cluster can change continuously for better localization of the target. In contrast to the static approach, members can belong to multiple clusters. The ongoing adaption of the cluster leads to a higher communication overhead.
  - Hybrid clusters combine features of the static and dynamic type. The goal is to combine simplicity and energy savings of static clusters with the flexibility and better localization-feature of dynamic clusters. After the deployment of the network, the system consists of static clusters. When an event occurs within the limits of a static cluster, it performs the fusion of the generated data itself. If an event affects two or more clusters, the affected static clusters are joined to a dynamic cluster. Creating dynamic clusters from static clusters is computationally less expensive than generate them completely dynamic.
- Tree-based [LPT06] [YWY10]: In a tree, the vertices are the sensor nodes, and the edges are the communication links. There exist many ways to establish a tree-based structure. One way would be to mark the sink node as the root. By merely flooding the network for a single time, the sink node can find the shortest

path to any reachable node. Another way would be to use a dynamic moving tree that transforms itself so that the node that detects the target and is closest becomes the tree's root.

- Face-based [HBLR05] [TCC07]: The network consists of multiple faces. A face is a group of nodes in a ring. A single node does not need the information of all nodes in the network. It is sufficient to collect data from nodes of adjacent faces. Face-based cooperation targets the problem of communication overhead in a cluster-based system triggered by cluster head selection and events that occur in overlapping clusters.

Target Localization

While localization is not a primary goal of target counting and is not a requirement for the basic functionality of that type of area monitoring, the target's position can be advantageous for components like energy management and node cooperation. For energy-based sensors, there is no additional effort needed to get the relative position of a target. The sensor is capable of recognizing the relative position of the target. In the case of binary sensors or numeric photo-electric sensors, it is different. Computational and communication overhead must be taken into account to estimate the placement of the targets. Target placement differs from target localization by not calculating the "exact" position but estimate between which sensors the target is located, without knowing how close the target is to a sensor. Like presented by Gandhi et al. [GKS08], target placement is more speculative than the "native" way of sensing the position of targets with energy-based sensors.

Energy Management

Due to limited battery capacity, energy management is a crucial topic in wireless sensor networks. We can differentiate between two classes of methods to increase energy-efficiency: sensing-related and communication-related methods. The first class exploits sensor reading to optimize future readings. Further, it applies filtering techniques to get precise information from noisy readings. The second class is responsible for electing active nodes, planning the sleep of nodes and dynamic clustering [DHA12].

Regarding the communication-related methods, only a subset of nodes has to participate in active communications. Either they are currently tracking a target or the target moves in their direction. The other nodes do not have to participate, and because it would be a waste of energy, they can go into sleep mode.

However, to know when the nodes need to be reawakened, there are strategies, which use the prediction results from the future-position prediction component. Based on the calculated trajectory and the speed of the target, relevant nodes are waked up on-demand [YW$^+$11].

3.1.2 Algorithms and applications

Different requirements need different solutions. Therefore, a large number of scientific applications were introduced in the past. This section will present two target counting algorithms and explain the most relevant principles to better understand this topic.

- The first one is the work of Gandhi et al. [GKS08]. They put much effort into analyzing target counting with a minimalistic sensing model to examine this fundamental problem's complexity and derive worst-case performance bounds. In particular, the minimalistic sensing model is a numeric counting model. It outputs (like already described in Target Detection) an integer value representing the number of distinct targets in the sensing range. A deliberate decision was made not to use the binary counting model because it is more suitable for tracking single targets. For multiple targets, it is just barely suitable under the restriction that the targets are pairwise widely-separated.

  So the counting sensor outputs the count of targets in the sensing range and gives no other information about the targets, such as the distance to the target. An example of a sensor with this sensing capability would be a low-cost radar sensor that can detect one or more targets but can not localize them. Other sensors that have these characteristics are, for instance, acoustic and infrared sensors.

  Based on the numeric counting model, Gandhi et al. [GKS08] stated that it is never possible to count precisely the number of targets. It is just possible to calculate upper and lower bounds on the target count. In a 1-dimensional setting, they could specify that no algorithm can count targets with an accuracy factor better than $\sqrt(2)$ if ideal sensing conditions are assumed. Target counting gets more accurate when minimizing the overlap of the sensor ranges.

  Sensor readings can also be noisy. This non-ideality has an impact on the accuracy of the results. This noise influences the sensor in a way that the sensor can output any value in a range (insert interval), where p is the noise and c is the real output of the sensor.

  Also, a 2-dimensional setting was considered. Unfortunately, in this case, it is not possible to define a worst-case accuracy factor like for the 1-dimensional counterpart. An approximation of the target count within factor $\sqrt(m)$ can be made if the maximum degree of overlap among sensor ranges is m.

- The second established algorithm is a compressive sensing-based approach. Zhang et al. [ZCZ+11] choose to employ CS because of the advances in sparse recovery for compressive sensing. One of their main goals was to prove the applicability of CS theory in target counting. They introduced a greedy matching pursuit algorithm (GMP), which can accurately recover a sparse signal with high probability. Compared to other well-known CS algorithms, GMP does not require any knowledge of the signal sparsity level. In addition to that, GMP is more computationally efficient. Another feature of the presented algorithm is that it can count and localize targets from multiple categories.

3.2 Simple Target Tracking

Target tracking is maybe the most basic use case in the field of area monitoring. Generally, the target detection system is responsible for tracking one or multiple objects in a predefined area and estimate the trajectories. Nodes are located in this area and can use different sensors to track objects moving in the controlled field. The coverage of each node is limited, so the nodes have to communicate to get an overview of the entire area.

The way the information goes through the system creates different formulations for a target-tracking problem: push-based, pull-based and guided-based. The most common formulation in target-tracking and the one we will focus on is the push-based principle. The push-based principle says that the nodes of a system push the information actively to a sink node. Conversely, nodes of a pull-based network send data to the sink node if it makes a query. In that case, the system acts like a distributed database. In the guided-based formulation, the system is responsible for keeping a so-called tracker up-to-date. A tracker can be a person or a vehicle that follows the computed trail to the target.

3.2.1 Components

Components useful for target counting also play an important role in target tracking. Within this section, we will describe slight variations to the target counting components. Furthermore, we introduce further ones like future-position prediction and target recovery.

Target Detection

As Vairo et al. [VACV10] described, a target is detected when a sensor captures information that matches the conditions that define an event. The detection process analyzes the sensor readings and decides if there is a target in the area.

Dependent on the environment conditions of the target tracking system, it uses different types of sensors. Sensor readings include infrared, acoustic, magnetic, light, radar, image and video data, to list the most common.

Like for target counting, there exist different sensing models that enable categorization of these sensor types:

- **Binary tracking:** Binary proximity sensors output a "1" when the target is in the sensing range, and "0" otherwise. This model's significant advantage offers a simple abstraction of the target tracking problem and can be applied to many applications, as stated out by Shrivastava et al. [SMS06]. Additionally, the communication costs are pretty low because of the 1-bit representation of sensor output.
- **Energy tracking:** The energy sensing model is based on the fact that targets emit energy signals such as acoustic, heat or light [WZLC14]. In addition to the binary sensing model, it provides supplementary information to the presence or absence of a target. Algorithms based on this more sophisticated sensing model take advantage of information like target velocity, target direction or distance to a target.

- **Compressive tracking:** Compressive Sampling (CS) can be seen as a subcategory of energy sensing. It enables a new way of data acquisition to reconstruct a sparse signal with a much lower sampling rate than the traditional Nyquist rate [WZLC14].

Besides the sensing model, the detection process of a sensor can use different algorithms to detect an event. To validate such algorithms, there exist detection models. With these detection models, the behaviour of an algorithm can be assessed considering different environmental conditions [NS10].

In the field of event-detection models, binary detection is the simplest and most popular one [WBS10] [HBK+10]. For a given event e, every sensor which distance d is smaller than the detection radius R detects the event e. This detection model evaluates algorithms under ideal conditions.

$$P(s,e) = \begin{cases} 1 & \text{if } d \leq R \\ 0 & \text{otherwise} \end{cases}$$

There are also other detection models, like the probabilistic detection models [HHYW11], which include parameters that represent the sensor technology and environmental factors and should clarify the non-ideality of real-world sensors. Further, a hybrid detection model that merges the binary and probabilistic detection model. [ZC04b]

Target Localization

This component has generally spoken one job: calculating the position of the target in the system. Initially, it is necessary to determine the distance between the detecting nodes and the target. This information can be provided by using the received signal strength indicator (RSSI) [XDD11] from energy-based sensors, which says nothing else than that the signal strength captured by a sensor decreases with the distance to the target. Assuming that the nodes know their location and the distance to the target, they can compute the target's position with trilateration or multilateration. There is also the bounding box technique, which uses squares instead of circles and is used to avoid floating-point calculations [BONL07, DOBNL08].

There also exist techniques without RSSI, presented by Wang [WBS10]. This technique works with binary sensors and assumes that the nodes have multiple neighbours whose sensing ranges intersect their sensing ranges. The target's location is always calculated when the target crosses the sensing range border of a node. The estimation becomes more precise when the crossed border is inside an intersection of two or more sensing ranges. So the nodes only know the position of a target, when it enters or leaves their sensing range. In that case, the distance is equal to the radius of the sensing range. If there is just one node involved, the target's position can be on the complete sensing range border. It is limited to a particular arc if the target crosses the boundary of neighbouring nodes.

Future-Position Prediction

For several reasons, it is essential to compute the future positions of the target [NLF07] [KS12]. The prediction component adapts network parameters like sampling rate, wakeup period and cluster size to the target's dynamic. The dynamics include the position, velocity and direction of the target. Standard estimation methods in sensor networks are the Kalman Filter, the alpha-beta filter (ABF) and the particle filter.

- **Kalman Filter:** One of the most popular methods to estimate future positions is the Kalman filter (KF), described by Welch and Bishop [WB$^+$95]. In general, KF estimates a process using a feedback loop of two steps, the Prediction-Step and the Correction-Step. In the Prediction-Step the current state is projected to a time step in the future. The Correction-Step is responsible for the feedback and corrects the estimations made in the past. The standard KF is intended to be used for systems that a linear model can describe. Variations for non-linear models are the Extended KF and the unscented KF.

- **Particle Filter:** Particle Filter (PF), described by Eberhart and Kennedy [KE95], is a recursive version of the sequential Monte Carlo Method. PF is an alternative to the Kalman Filter for applications with non-Gaussian noise. It is also often preferred if the system provides high computational power and is characterized by a slow sampling rate.

- **Alpha-Beta Filter:** The alpha-beta filter (a-b filter) is a derivative of the Kalman filter [Pen93]. The Kalman filter belongs to the best performing linear filter, but the excellent performance demands its price with complexity and intensive computation. The a-b filter simplifies processes of the Kalman filter and therefore reduces the complexity.

The standard versions of these prediction algorithms work in a centralized manner. To implement them in a distributed system, they have to be adapted. For instance, the Distributed Kalman Filter, explained by Olfati-Saber [OS07] and a distributed particle filter variation by Gu [Gu07].

Target recovery

As already mentioned, most target tracking applications try to extend the network lifetime and choose just a selected set of nodes to stay active. The other nodes are in sleep mode as long as the system predicts that the trajectory will cross the coverage area of these sleeping nodes. Depending on the speed of the target, the system wakes them up. However, the future-position prediction may be wrong and wakes up the wrong set of nodes. As a result, the target gets lost. Nevertheless, the lost can also have other reasons like sudden changes in the target's trajectory, the presence of obstacles in the monitored area or network errors.

Recovery mechanisms should help to recover the lost target. The challenge is to find the target precisely and fast with as few messages as possible. Therefore it is usually a trade-off between time and energy consumption. A fundamental recovery mechanism wakes up all nodes in the network. The drawback of this strategy is that it causes high energy consumption. Therefore, more complex solutions try to awaken a minimal amount of nodes to find the target. To give an example, Xu et al. [XWL04] and Yang and Sikdar [YS03] proposed a recovery mechanism that awakens the nodes around the predicated trajectory in case of a lost target. If the target is still not found, the mechanism awakens all nodes in the network. Of course, recovery mechanisms can be tailored precisely to the given problem of the target tracking application like in wildlife tracking. Lalooses et al. [LSC07a] apply a mechanism, which wakes up nodes around favourite places of the monitored animals.

Algorithms and applications

- **UWSN:** Multiple algorithms arose with a growing interest in underwater wireless sensor networks (UWSN). The applications range from Tsunami monitoring, over oilfield exploration projects to military purposes. Yu et al. [YLCS08] provided a distributed algorithm that tracks a single target with acoustic sensors. Therefore, it addresses issues regarding estimating the target position and reducing energy consumption by applying a Kalman filter. Energy efficiency is improved by putting currently not required nodes into sleep-mode.

  Another scheduling concept for underwater target tracking is presented by Hare et al. [HGS14], which uses dynamic clustering to establish an energy-efficient target-tracking sensor network. Hence, each node is modelled as a Probabilistic Finite State Automata that controls its sensing and communication activities. This mechanism wakes the nodes up if targets are present, and turns them off if they are absent.

- **Wildlife target tracking:** Wireless sensor networks can also support habitat and environmental monitoring. Some animal species are susceptible to human presence. Like explained by Anderson [And95], a group of scientists observed in a colony of seabirds that their daily visit of 15 minutes increased the egg and chick mortality rate by 20% in the year they monitored the colony.

  Sensor network monitoring offers an opportunity to observe animals without disrupting their normal animal behaviour. Lalooses et al. [LSC07b] studied the field of wildlife tracking and the mechanism of recovering a target during a wildlife monitoring operation. How to proceed if the target is lost? What steps should be taken to recover the lost target?

  Therefore, a recovery algorithm from Lalooses et al. [LSC07b] is purposed, taking advantage of hierarchical clustering. Every cluster selects a cluster head, which knows the radius of its monitored region. The parent of each cluster computes the radius of its area based on the information of its children. That mechanism is repeated until the root of the tree is reached. The recovery mechanism takes advantage of popular places for animals. Animals tend to visit these places because food and water are available. Other factors can also influence a place's popularity like a specific climate or some degree of protection from predators. Anyway, the system gathers information about the animals' behaviour over time and can derive the hot spots. In case that a target is lost, the system estimates the popular places that come into question because of time and distance from the last known position. Few sensors must be activated by restricting the new search area to the area around popular places, reducing the recovery strategy's energy consumption. Shortly after the system's initial deployment, the system cannot distinguish between unusual places and popular ones because it could not gather enough data about the monitored region. Therefore, it performs a fallback search using the maximum search area strategy. The search region is calculated by multiplicating the target's velocity and the time elapsed since the target was seen. All nodes in the calculated area are activated and perform a search.

3.3 Continuous Target Tracking

There are two types of targets in the field of area monitoring: simple and continuous targets. With target counting and target tracking, we treated the first type. However, that is not enough for some applications. It is necessary to track more complex targets than ones like animals, persons or vehicles. These targets can cover a larger field and are called phenomena or continuous targets because they are continuously distributed across a region. They are usually in three-dimensional space in reality, but in general, it is sufficient to know their locations and motions in a two-dimension plane. A fundamental characteristic of these targets is that they tend to diffuse and change their shape and size or split into smaller targets. Another challenge is when the continuous object spans a wide geographical area, and many sensors lie in this region. If each of these sensors had to communicate this target, it would lead to high overhead.

An efficient solution for this is to report only the boundary of the continuous object. That is not an easy task, and it can be quite challenging to track the border of a target like a wildfire or a toxic gas under the wind's influence.

Generally, for the most approaches in this area, it is desirable to minimize communication costs in operating the system. *Phenomena tracking* is closely related to target tracking (that also implies a close relation to target counting). It has to fulfil the same requirements as a target tracking system. Therefore, all components of an area monitoring algorithm from Section 2.1 are relevant for this tracking type. The principles of the components remain the same, so we will not go into detail again. We will explain the differences between simple and continuous target tracking by presenting two established algorithms.

The first one is CODA from Chang et al. [CLC08], a Continuous Object Detection and Tracking Algorithm. The main characteristic of this approach is that it uses a hybrid clustering technique. A more energy-efficient algorithm called PRECO (Predictive Tracking for Continuous Objects) is presented by Hong et al. [HNL$^+$10]. This algorithm aims to prolong the sensor network's lifetime by activating just the sensor nodes close to the continuous target.

Algorithms and applications

- **CODA:** The mechanism of CODA aims to track the boundaries of a moving target in the sensing area. This algorithm is based on Zha et al. [ZMK$^+$04] regarding the dynamic structuring of the network and introduces some improvements to reduce the overall communication costs.

  Within continuous target tracking, it is necessary to highlight a particular group of sensors, called the boundary sensors. The main characteristic of boundary sensors is that they detect an object in their vicinity but have one or more neighbours that do not detect an object. The boundary sensors estimate the location of the local boundary. Furthermore, they usually communicate these estimations to a sink node which calculates a global boundary. Therefore, some approaches exist, how the information is collected from the sinks. A very naive approach would be that the sinks collect all boundary sensors' information from one after another. That would lead to pretty high communication overhead.

  Zha et al. [ZMK$^+$04] introduced a strategy to counteract the high communication costs by grouping the boundary sensors into dynamic clusters. The local boundary information is fused within the corresponding cluster. Just the leader of the cluster communicates the fused boundary information to the sink. Thereby, less communication is required to compute the global boundary. With this principle of dividing the responsibility to smaller groups of nodes, the exchange of redundant information is narrowed down to a cluster, especially if there are just minor changes in the system that have to be communicated. Still, this technique can lead to many message exchanges since the boundary sensors are determined by requiring each sensor that detects a target to communicate with each one-hop neighbour to find out whether this sensor also noticed the same target. In addition to that, more clusters are needed when the continuous target grows in size. Therefore, new leaders of each cluster have to be elected. It is a fundamental problem for DCS (Dynamic Clustering Scheme) networks to find a sweet spot for the number of clusters and an efficient strategy for electing the leader.

  CODA is an extension of DCS and brings some advantages. It introduces a hybrid clustering scheme, as it hosts static and dynamic clusters to reduce communication overhead. The static clusters with a nominated leader are created during the initial deployment stage of the network. Any sensor which detects a target in its vicinity sends this information to his cluster leader. The vast amount of message exchange is not necessary anymore, as the leader identifies the boundary sensors. The boundary sensors become part of a dynamic cluster, where the leader is the same as for the static cluster. Therefore, an election mechanism is not necessary. Chang et al. [CLC08] describe the following steps:

  – *Static Clustering Scheme:* As already mentioned, the static clusters are constructed during the initial deployment stage with an already existing static clustering scheme like LEACH, TEEN, PEGASIS and so forth. During this stage, a leader for each cluster is nominated and acts as a local controller. The other nodes are responsible for sensing. Besides sensing, the leader fuses the data from the subordinated nodes and sends the result to the sink. Generally, it is assumed that every node is aware of its location. Within this step, the nodes of each static cluster are divided into inner and outer nodes. That distinction is essential for the next step.

- *Boundary Detection:* This step is responsible for determining how many clusters are affected by the continuous object and can be ensured by the outer nodes communicating with the one-hop outer nodes of other clusters. Additionally, the leader of an affected cluster knows which nodes have detected the object. CODA distinguishes between the following cases regarding the spreading of the continuous object:
    * Object concerns one static cluster.
    * Object crosses two static clusters.
    * Object crosses more than two static clusters.
    * Object affects all static clusters.
- *Boundary Recognition:* With the information of the step Boundary Detection the cluster leader can construct dynamic clusters out of the boundary sensors. Nodes can be added to or removed from the dynamic cluster depending on the continuous object's changes. The static cluster leader is also the dynamic cluster leader and transfers the fused data to the sink. On top of that, the sink determines the entire boundary of the object.
- *Boundary Tracking:* This task keeps track of the object boundary. The result should be that if the target moves, the dynamic cluster should be adjusted. Within CODA, there is the policy that only the nodes that detect diffusion of the object (those that see the object for the first time) or notice an object's disappearance are allowed to communicate with the leader. Thereupon, the steps Boundary Detection and Boundary Recognition are repeated.

- **PRECO:** CODA already tackles some issues of dynamic clustering schemes and the high amount of message exchanges that are needed:
    - Minimize communication cost through effective data delivery and aggregation
    - Reducing the number of reporting nodes

  That is achieved through the hybrid clustering mechanism.

  PRECO goes one step further and extends network lifetime by activating only sensors near the boundary of continuous objects and setting the other nodes to sleep mode. So to say, PRECO tries to find the minimum set of active nodes needed to keep the system alive and accurate. Moreover, the proposed algorithm predicts the future boundary line (border of the continuous object) by estimating the diffusing speed and direction of the current boundary line. With the predicted boundary line, the system activates sleeping nodes in the sensing range of the predicted boundary line. However, this is also accompanied by prediction errors that need to be dealt with. PRECO provides some wakeup mechanism to reduce the impact of prediction errors.

3.4 Decision tree for identification

This section aggregates the gathered information from the previous sections to support the identification of a given problem. Depending on the requirements of the area monitoring problem, our procedure suggests existing algorithms that should serve as a reference.

The first step of the procedure is to identify the general area monitoring pattern. Therefore, the question that should be asked is whether it is a counting or tracking problem. In the case of a tracking problem, we must distinguish between simple objects and phenomena.

So after identifying one of three possible patterns, we use one of the developed decision trees to continue the procedure.

- Target Counting: Figure 3.1
- Simple Target Tracking: Figure 3.2
- Continuous Target Tracking: Figure 3.3

We developed one decision tree for each area monitoring pattern we characterized in the previous sections. Since tracking is more complex than counting, the related trees have more decision nodes. However, it is quickly apparent that the decision nodes agree with the nice-to-have components of area monitoring. Furthermore, it is decisive whether one or more targets are observed simultaneously and influences the needed hardware and algorithm.

Our procedure matches the application requirements with the tree's decisions until a leaf is reached. Then the leaf suggests existing algorithms suitable for this kind of application. The applicability of the sensing models is discussed later in Section 3.5. The choice of the sensing model should be seen as the last step of the decision tree. Some decisions are not visible in our decision tree, like counting just one target, making no sense. However, there are also cases where we did not find a suitable algorithm for that configuration, like for the following simple target tracking configuration: Prediction? *No* $\rightarrow$ Target recovery? *Yes* $\rightarrow$ Multiple targets? *No*. Such configurations are excluded from the graph.

Algorithms listed with **bold** letters are already decentralized. Most of the algorithms listed with *italic* letters are distributed but not entirely decentralized. So there is any computation in a central point, and they have a single point of failure. However, that does not mean there is no decentralized solution for these algorithms. Moreover, some algorithms are purely centralized, so every sensing node communicates directly with the sink.

The elaborated node cooperation of each algorithm is indicated in the brackets. We have tree-based, cluster-based and face-based algorithms, but also ones with no additional concept of data aggregation. These algorithms are primarily supposed to work on a grid of sensor nodes. For that reason, we put a question mark in the brackets to indicate this assumption. If available, we also put a more familiar abbreviation of this algorithm into the brackets after the specification of the network structure.

Since this tree covers all possible decisions (also of the other pattern trees), we will explain the decision tree for simple target tracking in more detail. As displayed in Figure 3.2, the tree consists of totally four decisions. It depends on the given problem of area monitoring how sophisticated the solutions must be, and the decisions must fit the needs of the problem. The remarkable thing with the Prediction component in the decision trees is that it typically indicates an energy-saving mechanism. Prediction is used to estimate the target's future position to put sensing nodes into sleep or active mode. This mechanism is widely established and well known to extend the network's lifetime. Therefore, this component should be adequately considered when battery life plays a crucial role.

The next step is to identify the need for target recovery. In applications where losing targets can not be tolerated, a suitable strategy must be included in the algorithm to operate systematically in case of a target loss. In many algorithms, Target recovery is tightly coupled with the prediction of targets. Therefore, it is not a rarity that prediction results are used for the target recovery mechanism. So it is no surprise that we just found one algorithm that can recover targets but not predict them.

In OCO [TY06] we distinguish between sleeping and active nodes. The algorithm supposes that targets come from outside the sensing area, so when no target is currently tracked, just the border sensor nodes are active. When a sensor node tracks a target, it wakes up all of its neighbours to continue to track the target. So there is no sophisticated prediction of the target here. In case of a target loss, the active sensor node wakes up all network nodes to recover the track of the target again.

After deciding the Target recovery strategy, it must be defined if the system must track more than one target at a time. At this point, the later decided sensing model can not be neglected entirely, as it is not impossible but harder to track multiple targets with the binary sensing model and will lead to sacrifices in terms of accuracy.

We already mentioned that it is possible to transform a distributed algorithm into a decentralized one. An example for that transformation is shown in Section 4.6.3, where we used the gossip protocol to make the algorithm decentralized.

3.5 Applicability of the Sensing Models

Based on the sensing capabilities, state-of-the-art area monitoring algorithms are divided into four different categories: binary, numeric, energy and compressive sensing. As already mentioned, some wireless sensor network settings and their environment influence

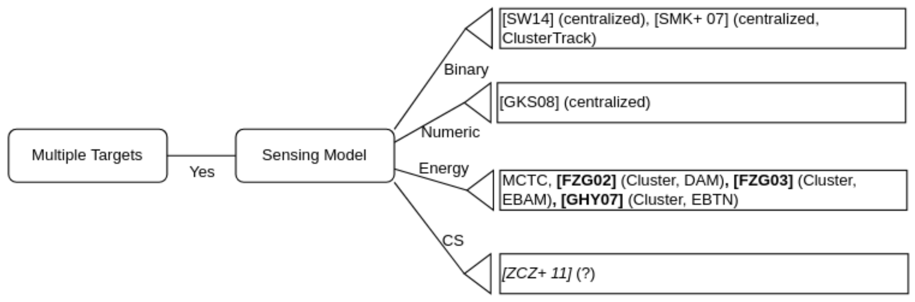

Multiple Targets
Yes
Sensing Model
Binary
Numeric
Energy
CS
[SW14] (centralized), [SMK+ 07] (centralized, ClusterTrack)
[GKS08] (centralized)
MCTC, [FZG02] (Cluster, DAM), [FZG03] (Cluster, EBAM), [GHY07] (Cluster, EBTN)
[ZCZ+ 11] (?)

Figure 3.1: Decision tree for target counting. It can be seen that counting is quite simple in comparison to tracking.

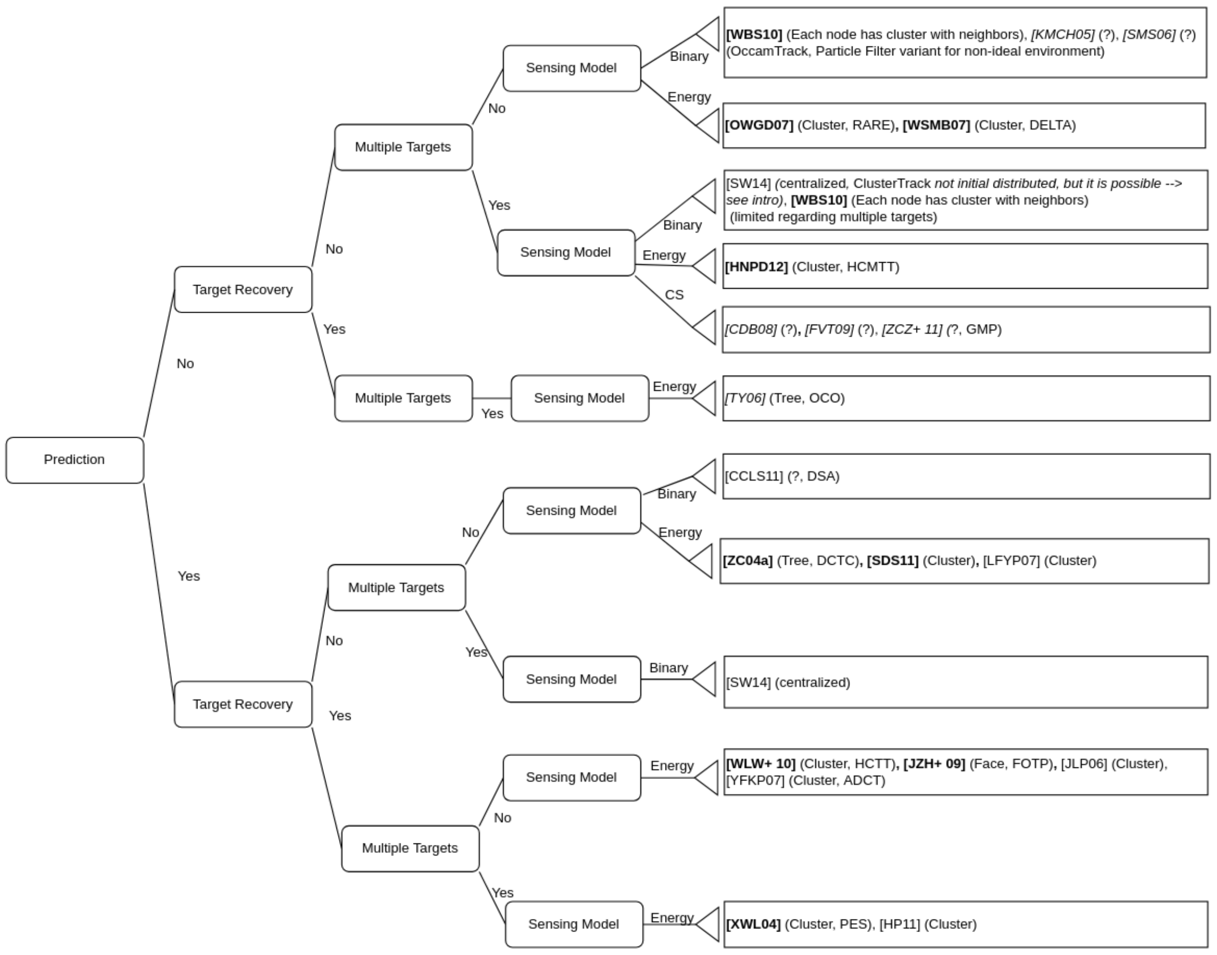

Prediction
No
Yes
Target Recovery
No
Yes
Multiple Targets
No
Yes
Sensing Model
Binary
Energy
[WBS10] (Each node has cluster with neighbors), [KMCH05] (?), [SMS06] (?) (OccamTrack, Particle Filter variant for non-ideal environment)
[OWGD07] (Cluster, RARE), [WSMB07] (Cluster, DELTA)
Sensing Model
Binary
Energy
CS
[SW14] (centralized, ClusterTrack not initial distributed, but it is possible --> see intro), [WBS10] (Each node has cluster with neighbors) (limited regarding multiple targets)
[HNPD12] (Cluster, HCMTT)
[CDB08] (?), [FVT09] (?), [ZCZ+ 11] (?, GMP)
Multiple Targets
Yes
Sensing Model
Energy
[TY06] (Tree, OCO)
Target Recovery
No
Yes
Multiple Targets
No
Yes
Sensing Model
Binary
Energy
[CCLS11] (?, DSA)
[ZC04a] (Tree, DCTC), [SDS11] (Cluster), [LFYP07] (Cluster)
Sensing Model
Binary
[SW14] (centralized)
Multiple Targets
No
Yes
Sensing Model
Energy
[WLW+ 10] (Cluster, HCTT), [JZH+ 09] (Face, FOTP), [JLP06] (Cluster), [YFKP07] (Cluster, ADCT)
Sensing Model
Energy
[XWL04] (Cluster, PES), [HP11] (Cluster)

Figure 3.2: Decision tree for tracking simple targets.

the different sensing methods' performance. Wu et al. [WZLC14] suggest to consider the following settings:

- Target Characteristics

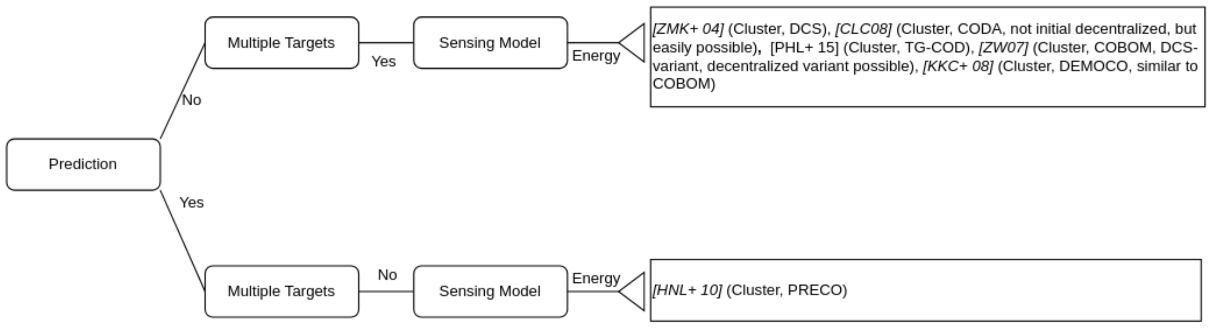

Prediction
No
Yes
Multiple Targets
Yes
Sensing Model
Energy
[ZMK+ 04] (Cluster, DCS), [CLC08] (Cluster, CODA, not initial decentralized, but easily possible), [PHL+ 15] (Cluster, TG-COD), [ZW07] (Cluster, COBOM, DCS-variant, decentralized variant possible), [KKC+ 08] (Cluster, DEMOCO, similar to COBOM)
Multiple Targets
No
Sensing Model
Energy
[HNL+ 10] (Cluster, PRECO)

Figure 3.3: Decision tree for tracking continuous targets.

- Target Deployment
- Sensor Deployment
- Communication Overhead

3.5.1 Target Characteristics

In general, there are two types of targets: visually distinguishable and visually indistinguishable targets. Numeric photo-electric sensors are exclusively able to count visually distinguishable targets but in return with highly accurate and continuously stable estimations [GHM$^+$10, WCXC12]. However, the working process of a numeric sensor is much more complicated and compute-intensive than in case of energy-based sensors, because it is necessary to execute a pattern recognition process in each numeric sensor. As a consequence, the maintenance of numeric sensors is more expensive than for energy-based sensors. On the contrary, readings of energy sensors are less stable than the readings of numeric sensors. The number of targets and the proximity of a target influences the sensing. Slightly changes in the position of a target can lead to a massive change in the reading of an energy sensor.

Nevertheless, energy-sensing is a compelling method for area monitoring since it can capture the presence of any target that emits certain kinds of energy, like light, heat or acoustic. Additionally, energy sensors can detect not only the presence of a target and the count of targets but also the relative position of a target.

Binary sensing models may be run with target contour information or energy information. Even though binary counting is stable and noise-resistant, it can provide a lower bound of the target count in most cases.

3.5.2 Target Deployment

Target Deployment addresses two aspects: target distribution and target density. These two factors may influence the sensing precision. Wu et al. [WZLC14] distinguish between 5 types of target distributions: uniform, piecewise uniform, hot-spot, multi hot-spot and mixed. It should be mentioned here that most algorithms assume a uniform distribution of the targets. In the case of non-uniform distribution, the counting precision decreases for these algorithms. However, some algorithms can handle a non-uniform target distribution, like the presented target counting method from Wu et al. [WCC+11] or Guo et al. [GHM+10]. These algorithms consider a piecewise uniform distribution.

Target density is a significant influencing factor for the performance of the different target sensing models. Because binary sensing models only consider sensors that can distinguish between two states, no matter how many targets are currently in the sensing region, this model should only be used for sparse targets. The compressive sensing model is also preferred to have sparse targets since the signal recovery algorithms are based on the assumptions that signals are sparse.

For energy-based sensing, it is necessary to consider the type of method for energy-sensing. If it is based on signal separation, the performance may decrease with a higher number of targets, as the signals can not be separated clearly. On the contrary, for algorithms based on signal landscape recovery and energy volume estimation, targets' density plays a tangential role.

For numeric sensing algorithms, the target density has little to no effect on the performance. These algorithms are based on local observations. As long as these local observations are precise, the aggregated result of these local measurements should also be accurate.

3.5.3 Sensor Deployment

Sensor deployment is related to a well-known and fundamental problem of wireless sensor networks: the coverage problem [FJ10, MA10]. Coverage is usually defined as measuring how well and how long the sensors can monitor an area of interest. Akshay et al. [AKHD10] divides coverage into three classes: area coverage, point coverage and barrier coverage. Area coverage deals with the problem of how to cover an area with sensors. In comparison to that, point coverage deals with the coverage of a defined set of interest points. Barrier coverage has the aim to decrease the probability of an undetected penetration of an area of interest.

However, our goal is to maximize the coverage percentage of an area. As a result, we have to focus on area coverage, which should also be seen as a minimization problem. The objective is to maximize coverage with as few nodes as possible. In particular, minimal overlapping with a maximization of the coverage is vital for binary and numeric sensing models. Otherwise, the estimation process becomes more complicated, and the sensing accuracy deteriorates.

For most protocols based on the energy model, the increase of sensor density does not significantly affect the sensing accuracy after reaching a certain level. Nevertheless, there are also energy-based protocols, like the DAM (Distributed Aggregate Management) protocol that was presented by Fang, Zhao and Guibas [FZG02], where more sensors help to differentiate between signals of different targets.

In compressive sensing, the minimum required signal sampling rate specifies the required number of sensors. More sensors would have little to no effect on the accuracy but would lead to redundant information processing.

3.5.4 Communication Overhead

In general, we can say that communication overhead depends on two parameters: the amount of information transmitted and the distance the information has to be communicated.

The binary sensing model offers the best communication overhead conditions because it only transmits 1 bit for state changes. Coupled with a decentralized algorithm, it is quite promising. In the case of energy counting models, algorithms based on DAM can be seen as wholly distributed, so the communication distance amounts to a minimum. Most other energy-based methods require sensors to transmit their states to a centralized server to get an aggregated result, which leads to a higher communication overhead. Although compressive counting follows in most cases a centralized strategy and the information transmitted may be complex, it can be considered as an economical communication model, as it saves communication costs by deploying just a small number of sensors.

To conclude the applicability of sensing models, we give (similar to Wu et al. [WZLC14]) an overview in Table 3.1 of the properties for each sensing model and setting.

Sensing model	Target Characteristics	Target Deployment	Sensor Deployment	Communication Overhead
Binary	Contour & Energy	Sparse	Counting: Sparse Tracking: Plentiful	Low
Numeric	Contour	Negligible	Sparse	High
Energy	Energy	Depends	Depends	Depends
CS	Energy	Sparse	Sparse	Low

Table 3.1: Table to highlight the properties of the sensing models for each setting

CHAPTER 4

Technical Framework for Area Monitoring

A contribution of this work is to provide a framework to ease the implementation of area monitoring applications using the gossip protocol. The framework should help to

- understand the characteristics of gossip protocols and how they have to be used to build a reliable area monitoring application
- speed up the development process
- create less error-prone solutions

The principles of gossip, presented in Section 2.3, are pretty straightforward, leading to many self-made solutions in the past. The problem with these custom solutions is that they are often error-prone and hardly adaptable because they were tailored for a specific use case. Therefore, we decided not to build the next custom gossip protocol solution to fill our requirements for area monitoring. Instead, we use the already established library MiCA, presented in Section 2.4. MiCA acts as our communication layer and takes over all tasks we expect from a basic gossip protocol solution. Hence, we can concentrate on the relevant parts of area monitoring without bothering with gossip.

The primary purpose of our work is to ease the implementation of new area monitoring solutions. Therefore, it provides a basic framework, which can be easily extended to fulfil a wide variety of requirements. In addition to that, it already offers functionality to simulate, test and visualize the implementation.

The following sections explain the main features of the framework. Moreover, we describe developing the framework and how the different examples we implemented influenced its core functions.

4.1 Event control

We introduced an interface `EventSource` in Listing 4.1 to decouple feeding the sensor nodes with events.

```
public interface EventSource {
   List<SensorEvent> getSensorData();
}
```

Listing 4.1: Depending on the implementation, sensors nodes are fed with input data. In real-world applications the data would be fetched from real sensors. In our simulations we get the data from YAML files.

That way, it is possible to use predefined input data for simulations and actual sensor data for real-world deployments. The reason to provide this opportunity is that it should be possible to feed simulations repeatedly with the same input data to review how changes in the implementation influence the system's output. In addition to that, evaluations can be made to check if gossip protocols are suitable for area monitoring and to show for which cases it fits better and for which not. However, such evaluations are more challenging if the input parameters change with every simulation run.

Due to the requirement of repeatedly feeding the simulations with the same input data, we decided to feed the sensor nodes of a simulation with structured and predefined data, in our case in the form of YAML (*Yet Another Markup Language*) files. With this method, there are no arbitrary events anymore. Instead, everything is written down in the YAML files and will determine what happens during the simulation.

```
nodes:
  0:
    - timeStampMillis: 20000
      targets:
        - x: 800
          y: 1000
          id: 1
    - timeStampMillis: 23500
      targets: []
  1:
    - timeStampMillis: 23500
      targets:
        - x: 800
          y: 1000
          id: 1
    - timeStampMillis: 27000
      targets:
        - x: 750
          y: 917
          id: 1
    - timeStampMillis: 30500
      targets:
        - x: 700
          y: 833
          id: 1
```

Listing 4.2: Example events for the sensor nodes with the id 0 and 1. Every node gets a list of events, which is triggered at the defined timestamp. It must be also defined, when the sensor node does not track a target anymore.

Like it can be seen in Listing 4.2, the events are written in a structured way and can be easily orchestrated. Another advantage of feeding the system with events that way is that it is also suitable for simulations where the sensor nodes are not on the same host.

4.2 Network Representation

In addition to the orchestration of sensor events, it is important for simulations and subsequent evaluations to define the network graph in a repeatable way, so every simulation run has the same prerequisites. The underlying library of our framework already provides a good base for this feature. More precisely, MiCA supports three network graphs:

- Random: The nodes in the graph have randomly selected neighbours.
- Complete: Every pair of distinct nodes is connected by an edge.
- Single ring: The graph nodes are linked to building a ring structure.

As we have the requirement to determine the graphs precisely for each simulation run, we implemented the framework feature to define a static network overlay in a YAML file, similar to feeding the system with events.

```
nodes:
   0:
      x: 100
      y: 100
      peers:
         - 1
         - 5
         - 6
   1:
      x: 300
      y: 100
      peers:
         - 0
         - 2
         - 6
         - 5
         - 7
```

Listing 4.3: Overlay configuration in YAML format for the nodes with id 0 and 1. Every node has coordinates and predefined neighbours.

Only two things have to be done to feed the simulation with a custom graph:

- Provide a project resource file called *static_overlay.yaml*, like the example in Listing 4.3
- Choose our added MiCA graph type option *custom*. In comparison, the standard MiCA graph types are *random*, *complete*, and *singlering*.

With these two actions, the communication graph is loaded from the provided YAML file, and it is possible to run tests on the same graph instance.

4.3 Logging

Logging is an essential aspect of software engineering, especially when there is a malfunction in the system and the engineer wants to analyze the deviation to the expected state of the system. Fortunately, MiCA already provides excellent logging features so that functionality could be used to establish expressive logs during the execution of our application. MiCA generates logs in the directory *mica_log*. Every line in the log represents an event and has the fields timestamp and address. Depending on the type of event, there are additional fields for that event. In our applications, we have to distinguish between three basic event types:

- **State:** This type of event describes the state of the node. That is done so that the protocol instance running on the node is serialized into JSON format. The advantage of serializing the protocol instance is that it is very flexible and independent of the implementation of the protocol, and it is machine-readable.
- **Select:** Records the communication partner of the gossip exchange
- **Rate:** Records the gossip rate. The default value for the gossip rate in MiCA is 1.

4.4 Visualization

Visualization can be helpful in different situations. On the one hand, problems in the system are found quicker, at least for operations visible to the user. On the other hand, graphic visualization also helps test the implementation and observe the communication and operations in the area monitoring application.

Therefore we integrated a way to visualize the running gossip protocols of the system smoothly. The visualization component reads the logs of a chosen node, deserializes them and feeds the library responsible for drawing the graph with selected data. We use GraphStream, a Java library for modelling dynamic graphs for drawing the graph.

Within the framework, we provide a basic implementation of the graph drawing application, which can be easily extended to support special needs and which runs separately from the respective target tracking application. The graph application has to run on the node with the information we want to visualize. The reason why one node is sufficient is strongly related to the principle of gossip, as we know that gossip protocols are convergently consistent.

In the graph application, we use the Apache Commons IO API included Tailer functionality, which is ideally suited to read log files because it reads files to the end and waits for new lines to be written into this file.

```java
public static void main(String[] args) {
    File file = new File(args[0]);
    TailerListener listener = new TailerListener(args[1]);
    Tailer.create(file, listener, 200);
}
```

Listing 4.4: The graph drawing application is independent of the main target tracking application and can be always deployed as a sidecar.

The main function of the graph application can be seen in Listing 4.4. It has two program arguments:

- The path to the log file
- The address of the node that should be supervised.

Furthermore, the TailerListener is listed in Listing 4.5:

```java
private static class TailerListener extends TailerListenerAdapter {
   @SneakyThrows
   @Override
   public void handle(String line) {
      Map<String, Object> map = objectMapper.readValue(line, new TypeReference<>() {});
      if (map.get("event_type").equals("mica-state-postupdate")) {
         MicaEvent event = objectMapper.readValue(line, MicaEvent.class);
         if (address.equals(event.getAddress())) {
            graphDrawer.drawGraph(event.getData().getState());
         }
      }
   }
}
```

Listing 4.5: `TailerListenerAdapter` provides an overrideable method `handle(String line)` to handle new lines in an observed file.

Because the log lines with event type *state* represent the current state of the gossip protocols and are serialized into JSON format, we want to deserialize them with the `ObjectMapper` class from the Jackson library. The read objects are then passed to the graph drawer class, using the GraphStream library to draw the sensor network and the targets. Example outputs of the graph drawing application can be seen in Figure 4.1.

4.5 Packaging

Packaging our Java application is just a minor feature, but we do not want to skip this feature here. With our build tool *Gradle* and the chosen structure of our project, it is easy to package the application simultaneously with two different configurations:

- **Simulation:** This package includes the simulation features to feed the system with custom events and a graph.
- **Real-world:** It includes the basic implementation without any simulation features to reduce the size of the binary and computation overhead for the real-world scenario.

These two packages can be combined with our visualization component shipped as a sidecar application. The visualization component is independent of the other packages and only needs a log file as input.

4.6 Reference Implementations

We distinguish between three main use cases in area monitoring: target counting, simple target tracking, and continuous target tracking. We provide a model solution for each of them. The solutions should help to get familiar with the framework and the underlying library MiCA. The node cooperation for the simple target tracking solution is tree-based. The algorithm for managing nodes and disseminating targets is based on OCO, introduced by Tran and Yang [TY06]. In comparison to that, the target counting and continuous target tracking solution works with cluster-based communication. The reference implementation of continuous target tracking is oriented on the mechanism of CODA [CLC08], presented in Section 3.3. The different network topologies make a massive difference on how the gossip protocols act. Therefore, we describe the different gossip protocols and the composition of the protocols to reach the goal of each use case.

4.6.1 Target counting

It is essential to understand that MiCA is designed to split work into smaller tasks and implement a gossip protocol for each of these tasks. So when we want to implement a target tracking application, we would split the whole application into the following smaller parts:

- Node cooperation
- Exchange of general information like node positions
- Exchange of targets

A significant advantage of MiCA is that it enables the composition of gossip protocols, so multiple protocols can efficiently communicate in parallel. Listing 4.6 shows the implementation of our reference solution for target counting.

```java
public class LayerStackCounting extends MergeCorrelated {
  private final StaticClusterOverlay staticClusterOverlay;
  private final TargetTrackingCluster targetTracking;
  private final LocalizationCluster localization;
  private final TargetCountSpreader aggregateSpreader;

  public LayerStackCounting(int nodeId, Overlay overlay, Point point, int clusterId, EventSource
       eventSource) {
    super();
    staticClusterOverlay = new StaticClusterOverlay(overlay, clusterId);
    targetTracking = new TargetTrackingCluster(staticClusterOverlay, eventSource);
    aggregateSpreader = new TargetCountSpreader(overlay, clusterId, targetTracking.getAggregate());
    localization = new LocalizationCluster(overlay, point, nodeId, staticClusterOverlay);
    setP1(
      MergeCorrelated.merge(MergeCorrelated.merge(staticClusterOverlay, localization),
      aggregateSpreader)
    );
    setP2(targetTracking);
  }
}
```

Listing 4.6: The target counting application is a composition of a bunch of protocols. Some protocols are input for other protocols, like the `StaticClusterOverlay` for the `TargetTrackingCluster`. In addition to that, the `EventSource` is injected to feed the sensor node with data.

The model solution of target counting has multiple layers of gossip protocols: `StaticClusterOverlay`, `LocalizationCluster`, `TargetTrackingCluster` and `TargetCountSpreader`. `LayerStackCluster` is responsible for the composition of the individual protocols. That is done either with a correlated merge or with an independent merge. Like in Section 2.3 already explained, we merge protocols in a correlated way if we want to force them to use the same communication partners for a gossip exchange (to reduce power consumption). If not, we use independent merging, and they will choose exchange partners independent from each other. `StaticClusterOverlay` divides the network into smaller groups of spatially adjacent nodes, also called clusters. For simplicity, we do not provide any sophisticated strategy to build clusters dynamically. Instead, each node knows during initialization to which cluster it is assigned. Therefore, the implementation of `StaticClusterOverlay` is pretty simple and is shown in Listing 4.7.

```
public class StaticClusterOverlay extends FailureDetector implements Overlay {
   @View
   public Overlay view;
   @Getter
   private int clusterId;
   @Getter
   private Set<Address> knownPeers = new HashSet<>();

   public StaticClusterOverlay(Overlay sourceOverlay, int clusterId) {
         super();
         this.view = sourceOverlay;
         this.clusterId = clusterId;
   }

   @Override
   public Distribution<Address> getView(RuntimeState runtimeState) {
         return Distribution.uniform(knownPeers);
   }

   @Override
   public void update(Protocol that) {
         super.update(that);
         StaticClusterOverlay other = (StaticClusterOverlay) that;

         compute(other);
         other.compute(this);
   }

   public void compute(StaticClusterOverlay other) {
         if (this.clusterId == other.clusterId) {
            knownPeers.add(other.getAddress());
         } else {
            knownPeers.remove(other.getAddress());
         }
   }
}
```

Listing 4.7: Protocol instances from the same cluster recognize each other. If two nodes are in the same cluster, they commit each other to be communication partners in the cluster overlay.

With a network divided into clusters, the protocol `TargetTrackingCluster` from Listing 4.6 is responsible for disseminating all targets inside the cluster. This dissemination follows an unstructured routine, where every node can communicate with all neighbours and exchanges all targets he detected, considering the principles of gossip. LocalizationCluster is a helper protocol that disseminates, among other things, the coordinates to tell the sink node where the sensor nodes are located. Finally, TargetCountSpreader distributes the target counts of each cluster throughout the network.

4.6.2 Simple target tracking

The model solution of simple target tracking (visualized in Figure 4.1a) uses another strategy for node cooperation. The application follows a tree-based structure and is implemented within four gossip protocols in Listing 4.8: MinAddressLeaderElection, SpanningTreeOverlay, LocalizationAggregator and TargetTrackingAggregator.

```java
public class LayerStack extends MergeCorrelated {
   private final MinAddressLeaderElection leaderElection;
   private final SpanningTreeOverlay tree;
   private final TargetTrackingAggregator targetTracking;
   private final LocalizationAggregator localization;

   public LayerStack(int nodeId, Overlay overlay, Point point, EventSource eventSource)
        {
         super();
         leaderElection = new MinAddressLeaderElection(overlay);
         tree = new SpanningTreeOverlay(leaderElection, overlay);
         localization = new LocalizationAggregator(tree, point, nodeId);
         targetTracking = new TargetTrackingAggregator(tree, eventSource);
         setP1(MergeCorrelated.merge(localization, targetTracking));
         setP2(MergeIndependent.merge(tree, leaderElection));
   }
}
```

Listing 4.8: The simple target tracking application consists of a variety of protocols that have to cooperate with each other. It differs from the cluster-based variant in that it spans a tree overlay over the network, so nodes are restricted to communicate with their children. The information flows upwards towards the root of the tree, which is in our solution the node with the least significant address.

MinAddressLeaderElection and SpanningTreeOverlay take care of the node cooperation. The former has to find a leader in the network (in our case, there is the simplification of choosing the node with the lowest address number), while the latter uses the leader election protocol to build up a tree topology and defines the found leader as the root of the tree. Thus, even though the tree overlay protocol needs the leader election protocol as input, these two protocols communicate independently and use a merge-independent strategy. With these two protocols, we provide a logical network topology used by the gossip protocols that will track targets. The nodes always communicate upwards to the root, acting as a sink node. Another vital feature of our overlay creating protocols is that it detects failures in the network. So if a node fails for any reason, the SpanningTreeOverlay protocol will exclude this node from the overlay and recreates a new tree.

LocalizationAggregator is more a helper protocol that disseminates the positions of the participating nodes to the root of the tree. That could be useful for several reasons, like energy management or improvements in creating the tree overlay. However, our reason for disseminating the positions of the nodes is that the sink node knows where the sensor nodes are located.

The primary gossip protocol in the application is `TargetTrackingAggregator`. It works on the generated rooted tree and aggregates the tracked targets to the root. An aggregator is a special case of a classic MiCA gossip protocol, as the data has to flow in a specific direction, and the data of both communication partners is aggregated.

```
public class TargetTrackingAggregator extends Aggregator<Set<Target>, Set<Target>> {
  @View
  public Overlay view;
  public Set<Target> aggregate = new HashSet<>();
  private final RootedTree tree;
  private final EventSource eventSource;

  public TargetTrackingAggregator(RootedTree tree, EventSource eventSource) {
    super(Protocol.Direction.PULL);
    this.tree = tree;
    this.view = tree.getChildrenAsOverlay();
    this.eventSource = eventSource;
  }

  @Override
  public Set<Target> getAggregate() {
    filterSummaries();

    List<SensorEvent> latestSensorEvents = eventSource.getSensorData();
    if (latestSensorEvents != null && !latestSensorEvents.isEmpty()) {
      aggregate.addAll(
        latestSensorEvents.stream()
          .flatMap(event -> event.getTargets().stream())
          .collect(Collectors.toList())
      );
    }
    getSummaries().values().forEach(aggregate::addAll);
    return aggregate;
  }
}
```

Listing 4.9: `TargetTrackingAggregator` is responsible to aggregate the tracked targets from the subtrees. In every gossip round, `getAggregate()` is called to fetch sensor readings of the current node with `eventSource.getSensorData()`. The own sensor readings are aggregated with the tracked targets of the children.

Listing 4.9 shows the implementation of `TargetTrackingAggregator`. The constructor defines that the parent always pulls the data from its children. The view of the parent node is limited to the children, so nodes can not choose a parent as an exchange partner. They always have to be chosen by their parents. The interface `EventSource` is just an abstraction of how sensors locate targets. The interface has one method with the signature `SensorEvent getSensorData()`. In the case of simulations, the event source is a YAML file.

The method that implements the aggregation of targets is `getAggregate()`. The MiCA update function calls it when a parent exchanges with a child node. Listing 4.10 shows that the aggregate of a child is stored in a map. The map's key is the address, and the value is the set of targets known to the child. Notice that the known targets represent all tracked targets by the child's subtree. Furthermore, `updatePull` is a special case for aggregators of the already presented method `public void update(Protocol that)` in Section 2.3.

```
public void updatePull(Aggregator<Summary, Aggregate> neighbor) {
   summaries.put(neighbor.getAddress(), neighbor.getSummary());
}
```

Listing 4.10: The `updatePull` method is a special case of `update()` and tailored for the Aggregator protocol. In comparison to the classic `update()`, information just flows in one direction, from the children to the parent.

With that strategy, tracked targets are passed to the root and can be fully accessed there.

The worst-case scenario regarding the logical network would be if the root failed. Then the network would need to find a new leader and rebuild the whole tree.

4.6.3 Continuous target tracking

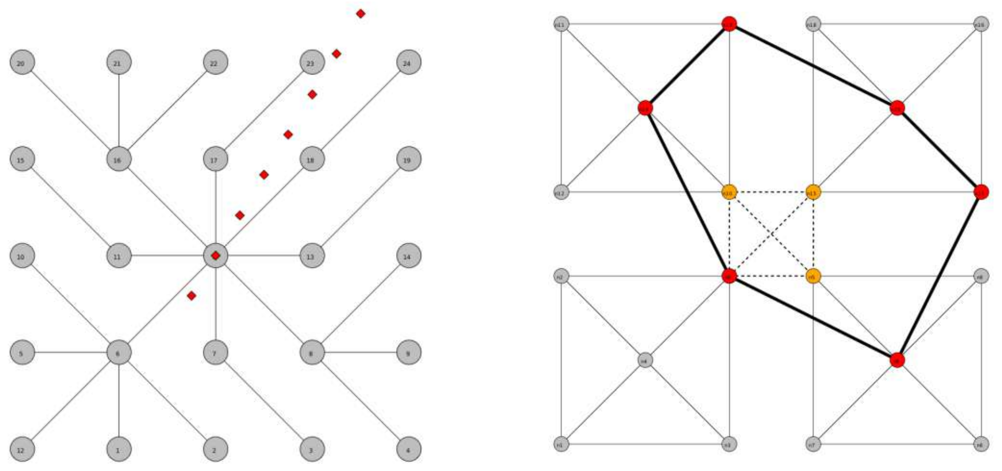
The continuous target tracking implementation (visualized in Figure 4.1b) is more complex than the previous model solutions regarding the number of gossip protocols involved. However, the composition of the protocols is similar to the previous ones. Therefore, we also reuse some gossip protocols from there.

```
public class LayerStack extends MergeIndependent {
  private NodeInformation nodeInformation;
  private StaticClusterOverlay staticCluster;
  private MinAddressLeaderElection leaderElection;
  private LeaderOverlay leaderOverlay;
  private Localization localization;
  private ContinuousTargetDetection continuousTargetDetection;
  private ConvexHullDissemination convexHullDissemination;

  public LayerStack(Address address, NodeInformation nodeInformation, Overlay overlay) {
    super();
    this.nodeInformation = nodeInformation;
    staticCluster = new StaticClusterOverlay(overlay, nodeInformation.getClusterId());
    leaderElection = new MinAddressLeaderElection(address, staticCluster);
    leaderOverlay = new LeaderOverlay(overlay, leaderElection);
    continuousTargetDetection = new ContinuousTargetDetection(staticCluster);
    localization = new Localization(staticCluster, nodeInformation);
    convexHullDissemination = new ConvexHullDissemination(leaderOverlay, localization,
         continuousTargetDetection, nodeInformation.getClusterId());
    setP1(
      MergeIndependent.merge(MergeCorrelated.merge(staticCluster, leaderOverlay),
      convexHullDissemination)
    );
    setP2(
      MergeCorrelated.merge(MergeCorrelated.merge(leaderElection, continuousTargetDetection),
      localization)
    );
  }
}
```

Listing 4.11: The `LayerStack` for the continuous target tracking application is a bit more complex and needs more protocols to provide the cluster-based topology.

In Listing 4.11 we work with six basic exchange gossip protocols, unlike the simple target tracking example, where we use the aggregator protocol. Furthermore, we want to reduce communication overhead by splitting the whole network into smaller clusters. For simplicity, we define which node is assigned to which cluster in advance, like in the



(a) Simple target tracking example

(b) Continuous target tracking example

Figure 4.1: Example outputs of our graph drawing application. Figure 4.1a shows the output of our simple target tracking application. All sensing nodes of the sytem are visible and a target crosses the sensing region. Figure 4.1b shows the output of our continuous target tracking application, which creates a convex hull around the phenomena.

reference implementation for target counting in Section 4.6.1. There would be other strategies in the real world, for instance, where the nodes decide in real-time, depending on their coordinates, which cluster they have to join. This strategy is advantageous if new nodes join the network and decide which cluster they join.

`StaticClusterOverlay` is responsible for forming the clusters in the network. Every protocol instance gets the cluster id as input, knowing which cluster it is assigned during instantiation. Thus, this gossip protocol aims to provide a logical network overlay for `ContinuousTargetDetection` for communication inside a cluster, mainly to exchange detected targets. The used sensors are binary, so they output true or false, whether they detect something or not.

In addition to that, each cluster elects a leader responsible for communicating with other clusters. Firstly, the `MinAddressLeaderElection` protocol selects the leader. Secondly, the `LeaderOverlay` protocol generates the overlay for communication between the leaders of each cluster.

As gossip protocols are convergently consistent, the leader gets all data from the participating nodes in his cluster over time. So he knows how far the continuous target is spreading. To reduce the information he has to communicate with other leaders, he generates a convex hull of all currently detecting nodes in his cluster. In our implementation, we use the Graham Scan [CLRS01] algorithm for finding the convex hull. That convex hull can be easily encoded in a single string and exchanged with the other leaders. Examples for encoding such a polyline can be found on the Google Maps Platform [goo].

The leaders share the convex hulls of their clusters combined to create an overall convex hull over all nodes in the network.

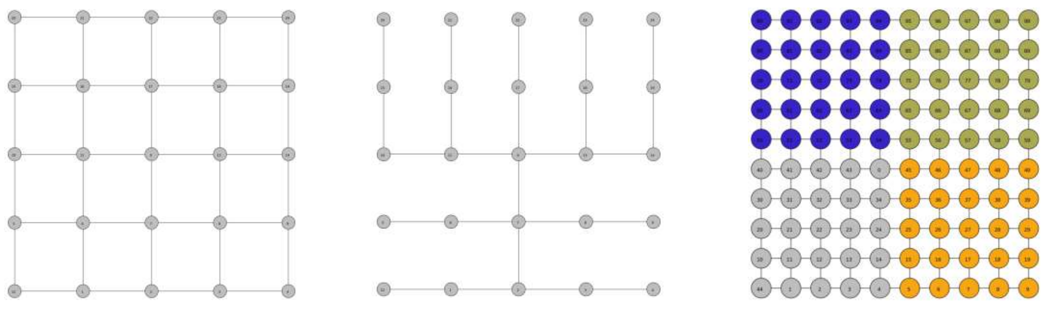
CHAPTER 5

Evaluation

This chapter deals with the applicability of gossip protocols for area monitoring. Of course, the decision about applicability comes with some requirements. However, we already know that gossip protocols can provide an energy-efficient, decentralized and fail-safe solution. The open question that still has to be clarified is whether data is fast enough disseminated in the network to keep track of the targets and inform the sink node in time about intruders or other important events. Obviously, it depends on the application requirements how fast events must be communicated in the network. For instance, in the case of a fire detection system in a forest, it can be pretty crucial to inform the fire brigade as fast as possible. On the contrary, this requirement could be relaxed in the case of a wildlife animal counting application where no immediate actions must be taken. Hereby, it can be sufficient that the sink node gets the target count of a specific sensor node within a few minutes.

The most influencing factor in the speed of disseminating events in the system is the network structure and the related node cooperation. Therefore, this chapter will compare three different approaches. The first one is a very naive approach, where every node can communicate with each neighbour and the events spread randomly through the unstructured network. This approach is closest to epidemic spreading and, therefore, to a "native" gossip protocol solution. In comparison to that, we also demonstrate a more sophisticated approach, where events are carried in a controlled way to the sink. Herefore, we use our tree-based simple target tracking implementation from Section 4.6.2. The third approach provides a cluster-based network structure, similiar to our model solution of target counting in Section 4.6.1, which can be seen as a more sophisticated variant of the naive approach. It combines the nodes into groups that operate autonomously and communicate the aggregates to the other clusters. In doing so, the amount of traffic communicated in the system can be drastically reduced.

We compare the different types of network structures by showing how long events travel from the sensor node to a sink. In the course of the comparison, we take the following significant input parameters into account:



(a) Naive Overlay (b) Tree Overlay (c) Cluster Overlay

Figure 5.1: The network structure of our different approaches. 5.1a shows the network of the naive approach. Every sensor node can communicate with each of the four neighbours. 5.1b shows the tree-based approach. In our tree-based application the parent pulls the information from the children. The root node is located in the middle. 5.1c represents a cluster-based layout with 100 nodes and a cluster size of 25 nodes.

- **Round length:** Since gossip protocols communicate periodically, the length of a gossip round significantly influences how long events need to reach the sink node. It defines the node exchange frequency.
- **Count of nodes:** The number of used nodes in a network leads logically to a higher number of hops till an event reaches the sink. Therefore it influences the time of travelling of events in the system.
- **Node degree:** Maximum number of edges a node can have. Since a node can communicate with one partner at a time (= gossip round), the increase of possible neighbours decreases the probability of each neighbour being selected.
- **(Target speed):** The target's speed does not influence how long it takes to communicate events to the network sink. Still, it is crucial to decide if a target tracking application makes sense with gossip communication. It depends on how fast the system should react to specific events.

We measured all evaluation results on a multicore platform with Intel Coffee Lake i7 8700K with 6 cores and 12 threads up to 4.7 GHz, 12 MB Intel Smart Cache, 16 GB of 3200 MHz DDR4 RAM, Java OpenJDK 11.0.10 Hotspot, operation system Manjaro Linux. Each simulated sensor node runs in its own Java thread, which is a native operation system thread. Our basic experimental setup simulates a 500 x 500 unit field with 25 sensor nodes. For simplicity, we assume that each sensor covers an area of 100 x 100 units. Each sensor node can communicate with the adjacent nodes. In the initial setup, we have a node degree of 4, so each node can have a maximum of four communication partners. The network setup is visualized with our frameworks visualization feature in Figure 5.1.

Firstly, we compare how the naive approach corresponds to the more sophisticated solutions with the given setup. All variants do the same tasks. The only difference is the network overlay and how the information is aggregated in the system. In our naive approach, the simplest method of gossiping is applied, and the information is spread randomly through the network until it eventually reaches the sink. In comparison to that, our tree-based simple target tracking solution takes the given network structure as input, agrees on a leader node and builds a spanning tree on top of it. Due to the resulting tree topology, every sensor node has a parent, and the flow of information goes towards the leader node, which acts as a sink. The cluster-based variant uses a very similar strategy as the naive approach. The information flow is also random, inside and outside a cluster. The only difference is that we control what data leaves the cluster, so just the aggregate is communicated to the nodes of the other clusters. The naive approach shares everything of every node "unfiltered" in the entire network.

To provide stable results in our simulations, generating a large amount of input data is necessary. Therefore, targets are simulated by generating a target event every five seconds on every sensor node in the system. With this way of providing input data, we accomplish that all nodes are included in our results, also the ones, that are furthest away from the sink. In addition, every sensor node provides a creation timestamp on each event, so when the event reaches the sink, it is possible to calculate the time needed to get to the sink.

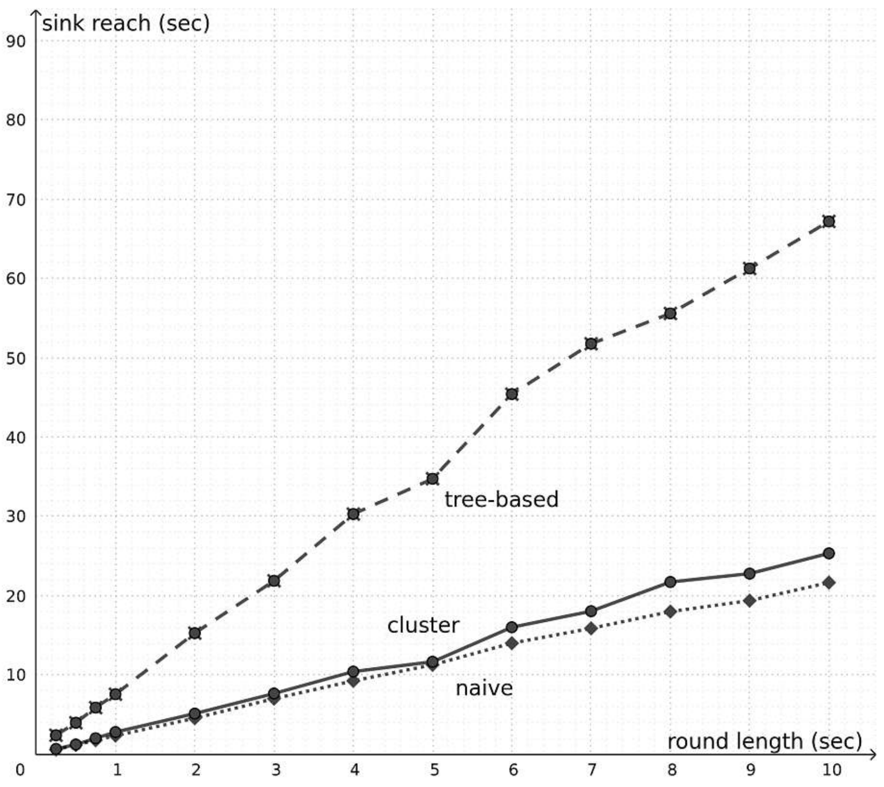


Figure 5.2: Average times for the naive, tree-based and cluster-based approach. The y-axis shows the average time in seconds how long the events needed to reach the sink. The x-axis represents the round length in seconds.

Figure 5.2 shows the average times the events needed to reach the sink within the naive, tree-based and cluster-based approach. In our simulations, we selected the node in the centre to be the sink node. We calculate the average from 10.000 samples for each run to get a reliable and robust average time. Also, we take different round lengths into account, up to ten seconds. The naive solution outperforms the tree-based solution in speed, even though the events have to do up to thirteen hops till they reach the sink. In comparison to that, the hops of an event in the tree-based approach are bounded by the tree's height (4 in our example). The main reason for the speed benefit of the naive approach is that this approach is better designed for the principles of the gossip protocol. The network structure provides more than one way a piece of information can move from the source to the sink, and the naive approach exploits that. Although the tree-based system is the more energy-efficient one, because less data is communicated in the network, data can take just one path. That can lead to higher delays since every subtree is randomly selected in a gossip round. Theoretically, the cluster-based approach combines the best of both variants in terms of speed and reducing communication overhead. It uses the same random dissemination strategy as the naive approach, both inside and outside a cluster. The nodes of a cluster share all the information they have, aggregate this data and just disseminate the aggregate to other clusters. So the cluster-based approach follows the principle that no information leaves the cluster that is not relevant for the others. Moreover, this approach also shows excellent results. However, we defined a fixed cluster size of 25 nodes and consequently, we operated in a single cluster in that simulation, so the results have to be taken with a grain of salt. Of course, we could have reduced the cluster size to 5 nodes. Nevertheless, on the one side, it would be an exaggeration to do so in a network setup with a total of 25 nodes. On the other side, the simulation with a single cluster should demonstrate that the cluster-based approach can compete with the naive one, even though we need an extra protocol for managing the cluster. Compared to the naive approach, we measured slightly worse times. As already mentioned, that can be justified because the cluster-based variant operates the same as the naive approach but needs an additional gossip protocol to manage the communication inside a cluster. Regardless, further simulations have up to 400 nodes and split the network into more than one cluster. We will see that the naive and cluster-based approach results remain similar in terms of time, as the only difference is the amount of data transmitted. The random spreading inside the network stays the same.

Furthermore, we compared the different approaches with higher numbers of sensor nodes but a fixed round length of 5 seconds. Figure 5.3 shows the results of the simulations with 25 up to 400 sensor nodes, which have (like before) a maximum of four communication partners. Again, we generated about 10.000 events for each run and took the average time an event needed to reach the sink. The naive and cluster-based approach is favourable in terms of speed and outperforms the tree-based one. Even though the results are very similar, it is evident that the aggregation inside a cluster reduces the communication overhead and leads to slightly lower dissemination times than the naive variant.

The next parameter that we want to evaluate is the node degree. Accordingly, we do simulations with a fixed number of 100 nodes and a round length of 5 seconds, but

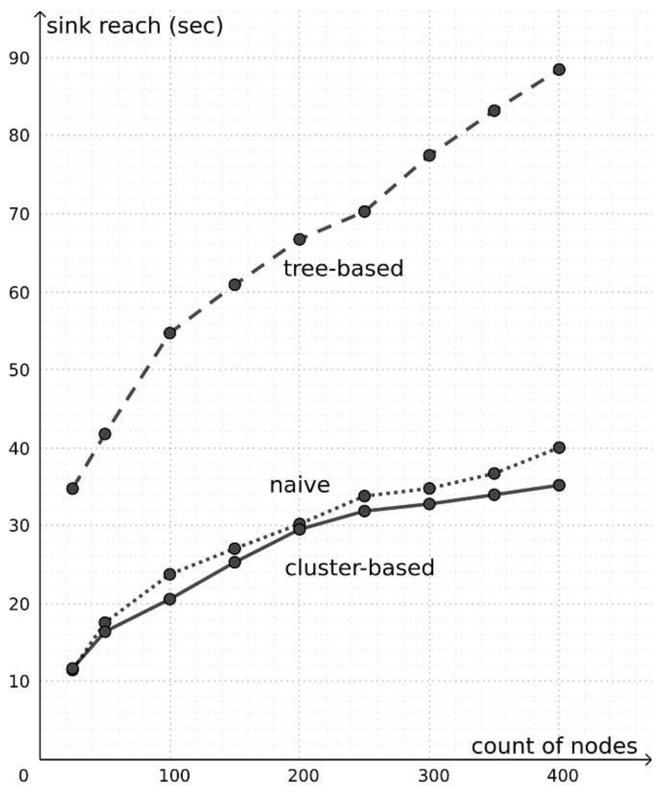


Figure 5.3: Average times for a fixed round length but varying node counts. In contrast to Figure 5.2, the x-axis represents the number of nodes

we vary the number of communication partners a node can have. To be more specific, Figure 5.4 shows the results of the different network structures with a node degree of 2, 4, 6 and 8 (these numbers represent all possibilities in our bidirectional network layout). Again, the naive and the cluster-based approach results are very similar, even though it is clear that the epidemic spreading benefits from a denser network and has better results with a higher node degree. However, the exception here is the tree-based solution that gets worse with a higher number of communication partners. It achieves the best result with a node degree of 4, then is nearly the same for 6, but gets even worse for a node degree of 8. The poorer outcomes may be associated with the characteristic of the tree-based approach that there is only one single path from each node to the sink. Therefore, the increase of possible neighbours decreases the probability of selecting a whole subtree. The node most affected by this is the sink node, shown in Figure 5.5. There we have a probability of 12.5% that the root selects a specific neighbour, and because exchange partners are randomly selected, that can take a long time, especially with a round length of 5 seconds.

The last input parameter we listed is the target's speed. This value does not influence the time needed to communicate an event to the sink. Still, it affects whether gossip

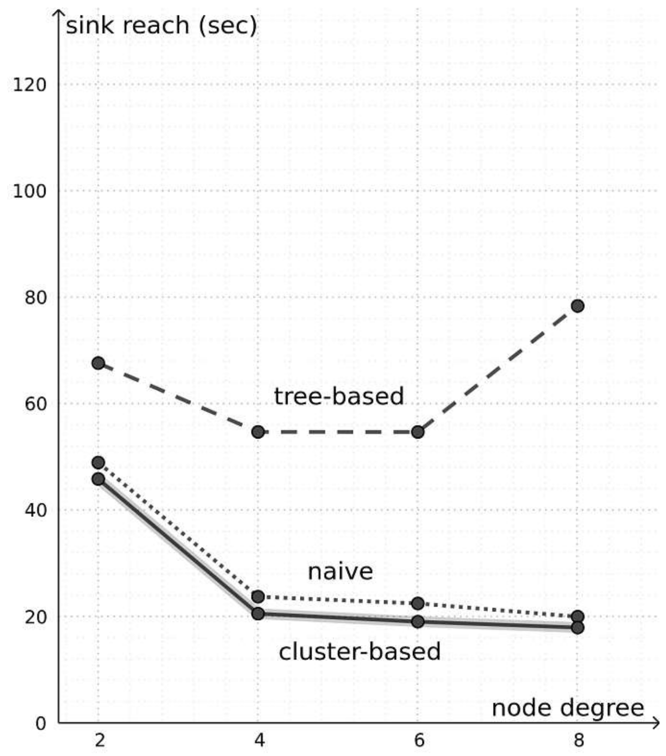


Figure 5.4: Average times for 100 nodes with a fixed round length but varying node degree. In contrast to Figure 5.2, the x-axis represents the number of communication partners a node can have

protocols are suitable for a specific type of application because they are supposed to react in time to particular events. An example of such an action would be to put nodes into sleep mode after recognizing that there is no target nearby these nodes. Another example would be calling the fire brigade after detecting a fire. If the target is faster than events reach the sink, the target could be out of range long before the system even react.

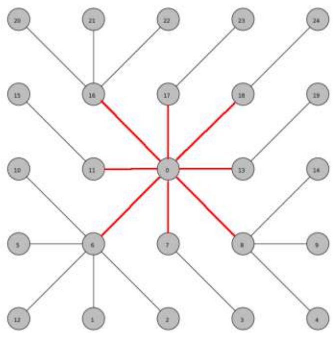


Figure 5.5: The tree-based approach with a higher node degree is problematic. In this example the sink node has 8 neighbours and therefore each of them a 12.5% probability to be selected from the root.

5.1 Reflection on the research questions

In the course of this thesis, we have supplied all the ingredients to answer the research questions from Section 1.3. Therefore, this section will revisit these questions and answer them concretely.

5.1.1 What are the main problem and application areas of area monitoring within the Internet of Things?

By surveying current scientific literature, we identified three common problems related to area monitoring for spatially-distributed Internet of Things: Target Counting, Simple Target Tracking and Continous Target Tracking. Hereby, the main challenges of implementing an area monitoring application are energy efficiency, accuracy and robustness.

Energy efficiency plays an essential role for resource-constrained devices and is indispensable for battery-powered sensor nodes. As communication is costly regarding energy consumption, an important concept in sensor networks is aggregating data to reduce the disseminated information. We presented different network structures that ease data aggregation, like tree-based and cluster-based networks. On the other hand, accuracy competes directly with energy efficiency, as higher accuracy results typically in higher energy consumption. In most cases, a trade-off has to be made between these requirements, why a perfectly accurate system cannot exist. We try to achieve robustness by providing a high-distributed or even a decentralized system. There should not be a single point of failure that can jeopardize the entire operation. Also, the system should be able to recover from communication and node failures.

We also introduced the essential and desirable components of area monitoring for the respective use cases. A fundamental prerequisite for all area monitoring applications is to choose a suitable sensing model. Furthermore, a node cooperation strategy can improve the system's overall performance and energy efficiency. In addition to that, target localization combined with energy management can save energy by putting inactive nodes into sleep mode. Finally, some components are just relevant for target tracking applications, like future-position prediction and target recovery.

5.1.2 How can we support engineers to develop area monitoring applications with gossip for the Internet of Things?

We support engineers with the following contributions:

- We gathered the knowledge from various scientific papers to provide the main characteristics of area monitoring and derive decision trees for finding existing algorithms depending on the requirements. It is also possible to elaborate unidentified requirements that were not considered.
- Furthermore, we presented the applicability of different sensing models to determine which model best supports the needs. The choice of the sensing model depends on different factors like target type, target deployment, sensor deployment. Also, the

communication overhead for a given sensing model plays an important factor. For instance, in the case of binary sensing, a single bit is sufficient to provide the state of a sensor node. Other sensing models need larger messages to share the state of a node.

- We implemented a framework on top of MiCA tailored to area monitoring solutions with gossip. It helps engineers by abstracting away low-level details to focus on the primary features of the application. In addition, the framework offers many features to ease the development of area monitoring applications. That is mainly accomplished by giving the engineers multiple tools to test their implementation. The framework supports mocked target events, custom network overlays, logging and visualization features. Further, we provide a model solution for each use case to demonstrate these features.

5.1.3 What is the effect of different network structures for area monitoring with gossip protocols?

We believe that the simulations in the previous section provide strong insights into how network structures influence the behaviour of an area monitoring system with gossip protocols. First, we observed that gossip protocols benefit from a dense network in terms of dissemination speed, so that information has many possibilities to travel from one node to another. Second, it is advantageous to have a method of aggregation in the system so that not every piece of information is shared with everyone in the network. Therefore, the overall communication overhead is reduced.

Based on these observations, we obtained promising results with the cluster-based approach because it combines the advantage of faster communication and energy efficiency by reducing the communication overhead in the network. Even though the naive variant had similar times to reach the sink in all simulations, it lacks a concept of aggregation and therefore shares everything with everybody. The network structure with the worst results regarding time to reach the sink was the tree-based structure. These results could be justified because the tree-based structure restricts communication ways in the network considerably, so it can not take full advantage of gossip.

In the end, the requirements determine which network structure should be selected. There will always be a trade-off between dissemination speed and energy efficiency. If communication costs are high and should be reduced to a minimum, the tree-based structure should be considered because data takes the shortest path to the sink. The only pain point regarding this network structure is that it takes, compared to the other structures, longer till the events reach the sink and should not be used for time-critical applications. An appropriate choice can be the naive approach in another scenario where communication costs are low, and only a small number of nodes are used. On the one hand, it is simpler to implement and on the other hand, time-critical applications can be considered. Suppose it is necessary to use a large number of nodes. In that case, an aggregation strategy should be considered to reduce energy consumption by not sharing every detail of information in the network.

CHAPTER 6

Conclusion & Future Work

Gossip protocols offer a competitive way of decentralized and energy-efficient communication for area monitoring and can be considered in many applications, where there is no demand to provide to-the-second accurate real-time information, and it is sufficient to synchronize the system's state within a few seconds. We distinguished in our simulations between three network structures that define the way of communication and aggregation in the system and have all their own right for different scenarios. By researching various scientific papers about area monitoring for spatially-distributed sensor nodes, we discovered three prominent patterns: Target Counting, Simple Target Tracking and Continuous Target Tracking. All area monitoring problems we got to know can be largely formulated in a related way to those three cases, even if they differ in specifics. With the help of the developed decision trees, we can support engineers to elaborate on these requirements and derive suitable existing algorithms that serve as a basis for further implementations. To prevent an error-prone hand-crafted solution, we developed a framework built on top of MiCA that provides different features to ease the implementation and testing of area monitoring applications. In addition to that, we provided reference implementations to help engineers get familiar with the framework by demonstrating its core features.

In future studies, experiments in a realistic setting can be conducted to spot possible problems when deploying gossip protocol solutions in real-world scenarios. Furthermore, we want to add, similar to the failure detection mechanism of MiCA, some more sophisticated target tracking features to our framework, like future-position prediction and target recovery. These complex features can follow different strategies depending on the requirements. We want to guarantee an easy integration of these features into an existing protocol composition and provide interchangeability of strategies. Moreover, it can be challenging to translate existing algorithms of these features into a gossip protocol variant. Another exciting topic for future work would be to use the gained knowledge from area monitoring with gossip and apply it to other domains of IoT.

List of Figures

3.1 Decision tree for target counting. It can be seen that counting is quite simple in comparison to tracking. 36
3.2 Decision tree for tracking simple targets. 36
3.3 Decision tree for tracking continuous targets. 37

4.1 Example outputs of our graph drawing application. Figure 4.1a shows the output of our simple target tracking application. All sensing nodes of the sytem are visible and a target crosses the sensing region. Figure 4.1b shows the output of our continuous target tracking application, which creates a convex hull around the phenomena. 51

5.1 The network structure of our different approaches. 5.1a shows the network of the naive approach. Every sensor node can communicate with each of the four neighbours. 5.1b shows the tree-based approach. In our tree-based application the parent pulls the information from the children. The root node is located in the middle. 5.1c represents a cluster-based layout with 100 nodes and a cluster size of 25 nodes. 54
5.2 Average times for the naive, tree-based and cluster-based approach. The y-axis shows the average time in seconds how long the events needed to reach the sink. The x-axis represents the round length in seconds. 55
5.3 Average times for a fixed round length but varying node counts. In contrast to Figure 5.2, the x-axis represents the number of nodes 57
5.4 Average times for 100 nodes with a fixed round length but varying node degree. In contrast to Figure 5.2, the x-axis represents the number of communication partners a node can have 58
5.5 The tree-based approach with a higher node degree is problematic. In this example the sink node has 8 neighbours and therefore each of them a 12.5% probability to be selected from the root. 58

List of Tables

3.1 Table to highlight the properties of the sensing models for each setting . . 39

Listings

2.1 Example implementation of a protocol with the high-level framework MiCA. The node that runs the protocol instance shares its current position and network related information like its current neighbours. 16
4.1 Depending on the implementation, sensors nodes are fed with input data. In real-world applications the data would be fetched from real sensors. In our simulations we get the data from YAML files. 42
4.2 Example events for the sensor nodes with the id 0 and 1. Every node gets a list of events, which is triggered at the defined timestamp. It must be also defined, when the sensor node does not track a target anymore. . 42
4.3 Overlay configuration in YAML format for the nodes with id 0 and 1. Every node has coordinates and predefined neighbours. 43
4.4 The graph drawing application is independent of the main target tracking application and can be always deployed as a sidecar. 44
4.5 `TailerListenerAdapter` provides an overrideable method `handle(String line)` to handle new lines in an observed file. 45
4.6 The target counting application is a composition of a bunch of protocols. Some protocols are input for other protocols, like the `StaticClusterOverlay` for the `TargetTrackingCluster`. In addition to that, the `EventSource` is injected to feed the sensor node with data. 46
4.7 Protocol instances from the same cluster recognize each other. If two nodes are in the same cluster, they commit each other to be communication partners in the cluster overlay. 47
4.8 The simple target tracking application consists of a variety of protocols that have to cooperate with each other. It differs from the cluster-based variant in that it spans a tree overlay over the network, so nodes are restricted to communicate with their children. The information flows upwards towards the root of the tree, which is in our solution the node with the least significant address. 48
4.9 `TargetTrackingAggregator` is responsible to aggregate the tracked targets from the subtrees. In every gossip round, `getAggregate()` is called to fetch sensor readings of the current node with `eventSource.getSensorData()`. The own sensor readings are aggregated with the tracked targets of the children. 49

4.10 The updatePull method is a special case of update() and tailored for the Aggregator protocol. In comparison to the classic update(), information just flows in one direction, from the children to the parent. 50
4.11 The LayerStack for the continuous target tracking application is a bit more complex and needs more protocols to provide the cluster-based topology. 50

Bibliography

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[AKHD10] Naregalkar Akshay, M Praveen Kumar, B Harish, and Sujata Dhanorkar. An efficient approach for sensor deployments in wireless sensor network. In *INTERACT-2010*, pages 350–355. IEEE, 2010.

[And95] John GT Anderson. Pilot survey of mid-coast maine seabird colonies: an evaluation of techniques. *Bangor, ME*, 12, 1995.

[Bir07] Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

[BONL07] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo F Nakamura, and Antonio AF Loureiro. Localization systems for wireless sensor networks. *IEEE wireless Communications*, 14(6):6–12, 2007.

[CCLS11] Jiming Chen, Kejie Cao, Keyong Li, and Youxian Sun. Distributed sensor activation algorithm for target tracking with binary sensor networks. *Cluster Computing*, 14(1):55–64, 2011.

[CDB08] Volkan Cevher, Marco F Duarte, and Richard G Baraniuk. Distributed target localization via spatial sparsity. In *2008 16th European Signal Processing Conference*, pages 1–5. IEEE, 2008.

[CLC08] Wang-Rong Chang, Hui-Tang Lin, and Zong-Zhi Cheng. Coda: A continuous object detection and tracking algorithm for wireless ad hoc sensor networks. In *2008 5th IEEE Consumer Communications and Networking Conference*, pages 168–174. IEEE, 2008.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 33.3: Finding the convex hull. *Introduction to Algorithms*, pages 955–956, 2001.

[CSN$^+$12] Andre N Campos, Efren L Souza, Fabiola G Nakamura, Eduardo F Nakamura, and Joel JPC Rodrigues. On the impact of localization and density control algorithms in target tracking applications for wireless sensor networks. *Sensors*, 12(6):6930–6952, 2012.

[CW08] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *IEEE signal processing magazine*, 25(2):21–30, 2008.

[DC08] Xiaojiang Du and Hsiao-Hwa Chen. Security in wireless sensor networks. *IEEE Wireless Communications*, 15(4):60–66, 2008.

[DHA12] Oualid Demigha, Walid-Khaled Hidouci, and Toufik Ahmed. On energy efficiency in collaborative target tracking in wireless sensor network: A review. *IEEE Communications Surveys & Tutorials*, 15(3):1210–1222, 2012.

[DOBNL08] Horacio Antonio Braga Fernandes De Oliveira, Azzedine Boukerche, Eduardo Freire Nakamura, and Antonio Alfredo Ferreira Loureiro. An efficient directed localization recursion protocol for wireless sensor networks. *IEEE Transactions on Computers*, 58(5):677–691, 2008.

[Duc13] Matt Duckham. *Decentralized spatial computing: foundations of geosensor networks*. Springer, 2013.

[FJ10] GaoJun Fan and ShiYao Jin. Coverage problem in wireless sensor network: A survey. *Journal of networks*, 5(9):1033, 2010.

[FVT09] Chen Feng, Shahrokh Valaee, and Zhenhui Tan. Multiple target localization using compressive sensing. In *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, 2009.

[FZG02] Qing Fang, Feng Zhao, and Leonidas Guibas. Counting targets: Building and managing aggregates in wireless sensor networks. *Palo Alto Research Center Technical Report*, 10298, 2002.

[FZG03] Qing Fang, Feng Zhao, and Leonidas Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *Proceedings of the 4th ACM International Symposium on Mobile ad hoc networking & computing*, pages 165–176, 2003.

[GHM$^+$10] Shuo Guo, Tian He, Mohamed F Mokbel, John A Stankovic, and Tarek F Abdelzaher. On accurate and efficient statistical counting in sensor-based surveillance systems. *pervasive and Mobile Computing*, 6(1):74–92, 2010.

[GHY07] Yan Guo, Bei Hua, and Lihua Yue. Energy-based target numeration in wireless sensor networks. In *Future Generation Communication and Networking (FGCN 2007)*, volume 2, pages 380–385. IEEE, 2007.

[GKS08] Sorabh Gandhi, Rajesh Kumar, and Subhash Suri. Target counting under minimal sensing: Complexity and approximations. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 30–42. Springer, 2008.

[goo] Encoded polyline algorithm format. `https://developers.google.com/maps/documentation/utilities/polylinealgorithm`. Accessed: 2021-12-14.

[Gu07] Dongbing Gu. Distributed particle filter for target tracking. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3856–3861. IEEE, 2007.

[HBK+10] Ting He, Chatschik Bisdikian, Lance Kaplan, Wei Wei, and Don Towsley. Multi-target tracking using proximity sensors. In *2010-MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*, pages 1777–1782. IEEE, 2010.

[HBLR05] Qingfeng Huang, Sangeeta Bhattacharya, Chenyang Lu, and Gruia-Catalin Roman. Far: Face-aware routing for mobicast in large-scale sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 1(2):240–271, 2005.

[HGS14] James Hare, Shalabh Gupta, and Junnan Song. Distributed smart sensor scheduling for underwater target tracking. In *2014 Oceans-St. John's*, pages 1–6. IEEE, 2014.

[HHYW11] Jau-Wu Huang, Chia-Mao Hung, Kai-Chao Yang, and Jia-Shung Wang. Probabilistic target coverage in wireless sensor networks. In *2011 International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 345–350. IEEE, 2011.

[HKB99] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, 1999.

[HNL+10] Seung-Woo Hong, Sung-Kee Noh, Euisin Lee, Soochang Park, and Sang-Ha Kim. Energy-efficient predictive tracking for continuous objects in wireless sensor networks. In *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1725–1730. IEEE, 2010.

[HNPD12] Faezeh Hajiaghajani, Marjan Naderan, Hossein Pedram, and Mehdi Dehghan. Hcmtt: Hybrid clustering for multi-target tracking in wireless sensor networks. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 889–894. IEEE, 2012.

[HP11] Yousef EM Hamouda and C Phillips. Adaptive sampling for energy-efficient collaborative multi-target tracking in wireless sensor networks. *IET wireless sensor systems*, 1(1):15–25, 2011.

[HSW⁺00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM Sigplan notices*, 35(11):93–104, 2000.

[JLP06] Guang-yao Jin, Xiao-yi Lu, and Myong-Soon Park. Dynamic clustering for object tracking in wireless sensor networks. In *International Symposium on Ubiquitious Computing Systems*, pages 200–209. Springer, 2006.

[JZH⁺09] Xu Ji, Yi-Ying Zhang, Sajjad Hussain, Dong-Xu Jin, Eun-Mook Lee, and Myong-Soon Park. Fotp: Face-based object tracking protocol in wireless sensor network. In *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pages 128–133. IEEE, 2009.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE, 2003.

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[KKC⁺08] Jung-Hwan Kim, Kee-Bum Kim, Sajjad Hussain Chauhdary, Wencheng Yang, and Myong-Soon Park. Democo: Energy-efficient detection and monitoring for continuous objects in wireless sensor networks. *IEICE transactions on communications*, 91(11):3648–3656, 2008.

[KMCH05] Wooyoung Kim, Kirill Mechitov, J-Y Choi, and Soo Ham. On target tracking with binary proximity sensors. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 301–308. IEEE, 2005.

[KS12] A Anand Kumar and Krishna M Sivalingam. Target tracking in a wsn with directional sensors using electronic beam steering. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*, pages 1–10. IEEE, 2012.

[LFYP07] In-Sook Lee, Zhen Fu, WenCheng Yang, and Myong-Soon Park. An efficient dynamic clustering algorithm for object tracking in wireless sensor networks. *Modeling, Control and Simulations*, 2007.

[LPT06] Chih-Yu Lin, Wen-Chih Peng, and Yu-Chee Tseng. Efficient in-network moving object tracking in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 5(8):1044–1056, 2006.

[LSC07a] Francine Lalooses, Hengky Susanto, and Chorng Hwa Chang. An approach for tracking wildlife using wireless sensor networks. In *The International Workshop on Wireless Sensor Networks (NOTERE 2007), IEEE, Marrakesh, Morocco*. Citeseer, 2007.

[LSC07b] Francine Lalooses, Hengky Susanto, and Chorng Hwa Chang. An approach for tracking wildlife using wireless sensor networks. In *The International Workshop on Wireless Sensor Networks (NOTERE 2007), IEEE, Marrakesh, Morocco*. Citeseer, 2007.

[Lyn96] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[MA10] Raymond Mulligan and Habib M Ammari. Coverage in wireless sensor networks: a survey. *Netw. Protoc. Algorithms*, 2(2):27–53, 2010.

[MISR11] Majdi Mansouri, Ouachani Ilham, Hichem Snoussi, and Cédric Richard. Adaptive quantized target tracking in wireless sensor networks. *Wireless Networks*, 17(7):1625, 2011.

[MLMKG06] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 123–132, 2006.

[MTP+19] Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger. Specification patterns for robotic missions. *IEEE Transactions on Software Engineering*, 2019.

[NLF07] Eduardo F Nakamura, Antonio AF Loureiro, and Alejandro C Frery. Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Computing Surveys (CSUR)*, 39(3):9–es, 2007.

[NS10] Eduardo F Nakamura and Efren L Souza. Towards a flexible event-detection model for wireless sensor networks. In *The IEEE symposium on Computers and Communications*, pages 459–462. IEEE, 2010.

[NSC+04] Silvia Nittel, Anthony Stefanidis, I Cruz, M Egenhofer, D Goldin, A Howard, Alexandros Labrinidis, Samuel Madden, Agnès Voisard, and M Worboys. Report from the first workshop on geo sensor networks. *ACM SIGMOD Record*, 33(1):141–144, 2004.

[OS07] Reza Olfati-Saber. Distributed kalman filtering for sensor networks. In *2007 46th IEEE Conference on Decision and Control*, pages 5492–5498. IEEE, 2007.

[OWGD07] Elizabeth Olule, Guojun Wang, Minyi Guo, and Mianxiong Dong. Rare: An energy-efficient target tracking protocol for wireless sensor networks. In

2007 International Conference on Parallel Processing Workshops (ICPPW 2007), pages 76–76. IEEE, 2007.

[PCJ+14] Lonnie Princehouse, Rakesh Chenchu, Zhefu Jiang, Kenneth P Birman, Nate Foster, and Robert Soulé. Mica: A compositional architecture for gossip protocols. In *European Conference on Object-Oriented Programming*, pages 644–669. Springer, 2014.

[Pen93] Robert Penoyer. The alpha-beta filter. *C User's Journal*, 11(7):73–86, 1993.

[PHL+15] Soochang Park, Seung-Woo Hong, Euisin Lee, Sang-Ha Kim, and Noel Crespi. Large-scale mobile phenomena monitoring with energy-efficiency in wireless sensor networks. *Computer Networks*, 81:116–135, 2015.

[SCN11] Efren Lopes Souza, Andre Campos, and Eduardo Freire Nakamura. Tracking targets in quantized areas with wireless sensor networks. In *2011 IEEE 36th Conference on Local Computer Networks*, pages 235–238. IEEE, 2011.

[SDS11] Surendra Sharma, Sarang Deshpande, and Krishna M Sivalingam. Alpha-beta filter based target tracking in clustered wireless sensor networks. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–4. IEEE, 2011.

[SMK+07] Jaspreet Singh, Upamanyu Madhow, Rajesh Kumar, Subhash Suri, and Richard Cagley. Tracking multiple targets using binary proximity sensors. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 529–538, 2007.

[SMS06] Nisheeth Shrivastava, R Mudumbai U Madhow, and Subhash Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 251–264, 2006.

[SNP16] Éfren L Souza, Eduardo F Nakamura, and Richard W Pazzi. Target tracking for sensor networks: A survey. *ACM Computing Surveys (CSUR)*, 49(2):1–31, 2016.

[SS05] Yi Shang and Hongchi Shi. Coverage and energy tradeoff in density control on sensor networks. In *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, volume 1, pages 564–570. IEEE, 2005.

[SW14] Lei Song and Yongcai Wang. Multiple target counting and tracking using binary proximity sensors: Bounds, coloring, and filter. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 397–406, 2014.

[TCC07] Hua-Wen Tsai, Chih-Ping Chu, and Tzung-Shi Chen. Mobile object tracking in wireless sensor networks. *Computer communications*, 30(8):1811–1825, 2007.

[TY06] SP Manh Tran and T Andrew Yang. Oco: Optimized communication & organization for target tracking in wireless sensor networks. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06)*, volume 1, pages 428–435. IEEE, 2006.

[VACV10] Claudio Vairo, Giuseppe Amato, Stefano Chessa, and Paolo Valleri. Modeling detection and tracking of complex events in wireless sensor networks. In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 235–242. IEEE, 2010.

[WB+95] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter, 1995.

[WBS10] Zijian Wang, Eyuphan Bulut, and Boleslaw K Szymanski. Distributed energy-efficient target tracking with binary sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 6(4):1–32, 2010.

[WCC+11] Dengyuan Wu, Xiuzhen Cheng, Dechang Chen, Wei Cheng, Biao Chen, and Wei Zhao. A monte carlo method for mobile target counting. In *2011 31st International Conference on Distributed Computing Systems*, pages 750–759. IEEE, 2011.

[WCXC12] Dengyuan Wu, Dechang Chen, Kai Xing, and Xiuzhen Cheng. A statistical approach for target counting in sensor-based surveillance systems. In *2012 Proceedings IEEE INFOCOM*, pages 226–234. IEEE, 2012.

[WD04] Michael F Worboys and Matt Duckham. *GIS: a computing perspective*. CRC press, 2004.

[WLW+10] Zhibo Wang, Wei Lou, Zhi Wang, Junchao Ma, and Honglong Chen. A novel mobility management scheme for target tracking in cluster-based sensor networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 172–186. Springer, 2010.

[WSMB07] Markus Wälchli, Piotr Skoczylas, Michael Meer, and Torsten Braun. Distributed event localization and tracking with wireless sensors. In *International Conference on Wired/Wireless Internet Communications*, pages 247–258. Springer, 2007.

[WZLC14] Dengyuan Wu, Bowu Zhang, Hongjuan Li, and Xiuzhen Cheng. Target counting in wireless sensor networks. In *The art of wireless sensor networks*, pages 235–269. Springer, 2014.

[XDD11] Enyang Xu, Zhi Ding, and Soura Dasgupta. Target tracking and mobile sensor navigation in wireless sensor networks. *IEEE Transactions on mobile computing*, 12(1):177–186, 2011.

[XL07] Yingqi Xu and Wang-Chien Lee. Compressing moving object trajectory in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 3(2):151–174, 2007.

[XWL04] Yingqi Xu, Julian Winter, and Wang-Chien Lee. Prediction-based strategies for energy saving in object tracking sensor networks. In *IEEE International Conference on Mobile Data Management, 2004. Proceedings. 2004*, pages 346–357. IEEE, 2004.

[YFKP07] WenCheng Yang, Zhen Fu, JungHwan Kim, and Myong-Soon Park. An adaptive dynamic cluster-based protocol for target tracking in wireless sensor networks. In *Advances in Data and Web Management*, pages 157–167. Springer, 2007.

[YLCS08] Chang Ho Yu, Kang Hoon Lee, Jae Weon Choi, and Young Bong Seo. Distributed single target tracking in underwater wireless sensor networks. In *2008 SICE Annual Conference*, pages 1351–1356. IEEE, 2008.

[YS03] Haiming Yang and Biplab Sikdar. A protocol for tracking mobile targets using sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, pages 71–81. IEEE, 2003.

[YW$^+$11] Dongmei Yan, Jinkuan Wang, et al. Sensor scheduling algorithm target tracking-oriented. *Wirel. Sens. Netw.*, 3(8):295–299, 2011.

[YWY10] Li-Hsing Yen, Bang Ye Wu, and Chia-Cheng Yang. Tree-based object tracking without mobility statistics in wireless sensor networks. *Wireless Networks*, 16(5):1263–1276, 2010.

[ZC04a] Wensheng Zhang and Guohong Cao. Dctc: dynamic convoy tree-based collaboration for target tracking in sensor networks. *IEEE Transactions on wireless communications*, 3(5):1689–1701, 2004.

[ZC04b] Yi Zou and Krishnendu Chakrabarty. Sensor deployment and target localization in distributed sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(1):61–91, 2004.

[ZCZ$^+$11] Bowu Zhang, Xiuzhen Cheng, Nan Zhang, Yong Cui, Yingshu Li, and Qilian Liang. Sparse target counting and localization in sensor networks based on compressive sensing. In *2011 Proceedings IEEE INFOCOM*, pages 2255–2263. IEEE, 2011.

[ZGG04] Feng Zhao, Leonidas J Guibas, and Leonidas Guibas. *Wireless sensor networks: an information processing approach*. Morgan Kaufmann, 2004.

[ZMK^{+}04] Hongyuan Zha, JJ Metzner, George Kesidis, et al. Dynamic cluster structure for object detection and tracking in wireless ad-hoc sensor networks. In *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*, volume 7, pages 3807–3811. IEEE, 2004.

[ZW07] Cheng Zhong and Michael Worboys. Energy-efficient continuous boundary monitoring in sensor networks. In *Technical Report*. Citeseer, 2007.