ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
MONITORAGGIO E GESTIONE DELLE STRUTTURE E DELL'AMBIENTE - SEHM2

Ciclo 35

Settore Concorsuale: 01/B1 - INFORMATICA
Settore Scientifico Disciplinare: INF/01 - INFORMATICA

Hybrid Ground-Aerial Mesh Networks for IoT Monitoring Applications: Network Design and Software Platform Development

Presentata da:
Leonardo Montecchiari

Coordinatore Dottorato
Luca De Marchi

Supervisore
Marco di Felice

Co-supervisore
Angelo Trotta

Esame finale anno 2023

Alla famiglia di via pinto

To Giulia

To the Prism Lab

To all of my supporters

Abstract

The Internet of Things (IoT) has grown rapidly in recent years, leading to an increased need for efficient and secure communication between connected devices. Wireless Sensor Networks (WSNs) are composed of small, low-power devices that are capable of sensing and exchanging data, and are often used in IoT applications. In addition, Mesh WSNs involve intermediate nodes forwarding data to ensure more robust communication. The integration of Unmanned Aerial Vehicles (UAVs) in Mesh WSNs has emerged as a promising solution for increasing the effectiveness of data collection, as UAVs can act as mobile relays, providing extended communication range and reducing energy consumption. However, the integration of UAVs and Mesh WSNs still poses new challenges, such as the design of efficient control and communication strategies.

This thesis explores the networking capabilities of WSNs and investigates how the integration of UAVs can enhance their performance. The research focuses on three main objectives: (1) Ground Wireless Mesh Sensor Networks, (2) Aerial Wireless Mesh Sensor Networks, and (3) Ground/Aerial WMSN integration. For the first objective, we investigate the use of the Bluetooth Mesh standard for IoT monitoring in different environments. The second objective focuses on deploying aerial nodes to maximize data collection effectiveness and QoS of UAV-to-UAV links while maintaining the aerial mesh connectivity. The third objective investigates hybrid WMSN scenarios with air-to-ground communication links. One of the main contribution of the thesis consists in the design and implementation of a software framework called "Uhura", which enables the creation of Hybrid Wireless Mesh Sensor Networks and abstracts and handles multiple M2M communication stacks on both ground and aerial links. The operations of Uhura have been validated through simulations and small-scale testbeds involving ground and aerial devices.

Table of contents

List of figures

List of tables

List of Algorithms

Chapter 1

Introduction

Thanks to the recent advances in technology and the decreasing cost of hardware solutions, in the last decade, the use of the Internet of Things(IoT) has grown exponentially [1]. It is being used in many different applications, e.g., smart home, smart agriculture, and smart grid, and usually involves a certain number of devices connected to accomplish a specific task. Each device is considered part of the same system since they are connected to the same network, like sensors deployed in a field to monitor the humidity and temperature of the soil or cameras installed in a building to monitor the occupancy.

The increasing usage of IoT has resulted in a growing need for efficient and secure communication between these devices. One of the key challenges in IoT is to ensure reliable networking, especially in situations where the devices are distributed in a rural area, have limited resources, or are subject to harsh environments. This is where wireless sensor networks (WSNs) come into play [2].

WSNs are composed of a large number of small, low-power devices, such as sensors, that are capable of sensing, computing, and exchanging data. They are often used to collect data from the environment and transmit it to a central device, called a sink, for data processing or simply storage. Mesh WSN or WMSN is a type of network to address these challenges [3]. It involves using intermediate nodes to forward data from a source to a destination, allowing for more robust and efficient communication. In Structural Health Monitoring (SHM) applications, mesh networking is particularly useful as it collects data from various sensors distributed throughout a large-scale structure and enables the data transmission to a sink unit for analysis and monitoring of the structural integrity [4].

However, the use of mesh networking in SHM also raises new challenges, such as the need for efficient data collection algorithms to enable the real-time monitoring of scenario. Additionally, the design of energy-efficient communication protocols is crucial to ensure the lifespan of the sensor devices, which are often battery-powered. Unmanned Aerial Vehicle-

aided Wireless Sensor Networks (UAV-aided WSNs) represent a promising solution for dealing with these challenges [5, 6]. In this approach, UAVs act as mobile relays for the sensor nodes, providing an extended communication range and reducing the energy consumption of the devices. Furthermore, UAVs can be deployed in areas where it is difficult or impossible to get access with traditional methods, such as remote locations or hazardous environments. Multiple UAVs, or, in general, unmanned vehicles, have different communication capabilities compared to WSNs, such as higher mobility, larger communication range, and the ability to provide a relay between different WSNs.

At the same time, the integration of UAVs in WSNs also poses new challenges, such as the design of efficient control and communication strategies to ensure the coordination and cooperation of the UAVs with the ground devices. Indeed, in a swarm of UAVs or ground vehicles (UGVs), nodes can autonomously coordinate their activities and cooperate to accomplish a given task. Due to unpredictable environmental conditions, wireless communication on air-to-air, ground-to-air, and ground-to-ground links can experience completely different channel conditions. For this reason, several Machine-to-Machine (M2M) communication technologies have been proposed with different Quality of Service (QoS) characteristics in terms of range, bandwidth, and energy consumption profile [7]: such fragmentation poses formidable challenges on the how to support the joint utilization of multiple M2M stacks in heterogeneous IoT robotic environments.

In summary, the primary objective of this thesis is to explore the networking capabilities of WSNs for IoT monitoring applications and investigate how the integration of UAVs can enhance their performance. Specifically, the research focuses on three main objectives: (1) design and deployment of solutions enabling the creation of Ground Wireless Mesh Sensor Networks (WMSN) solutions, (2) design and deployment of solutions enabling the creation of Aerial Wireless Mesh Sensor Networks, and (3) Ground/Aerial mesh integration in the so-called Hybrid WMSN. The first objective aims to investigate the use of the emerging Bluetooth Mesh [8] standard for IoT monitoring applications in different environments, including outdoor and indoor settings. The second objective focuses on deploying aerial networks so that they can maximize the effectiveness of the data collection in terms of ground nodes connected and in QoS of the mesh links while preserving the aerial mesh connectivity. Finally, the third objective is to investigate hybrid scenarios with air-to-ground communication links among the two network meshes. For objectives (2) and (3), a major contribution of the thesis consisted in the design and implementation of *Uhura*, a novel software framework that enables multi-radio H-WMSN, abstracting and handling multiple M2M communication stacks in both ground and aerial nodes.

The thesis is structured into five main chapters. Chapter 2 provides an overview of the related work in the field of WSNs for SHM applications, Wireless Mesh Sensors Networks, and UAV-aided sensor networks. Chapter 3 clarifies the scenario and the objectives of the thesis, including the methodologies adopted. Chapter 4 investigate the Bluetooth Mesh performance as a promising Wireless technology for Ground WMSN development in IoT scenarios. Chapter 5 describes our contribution on aerial WMSN depployment; more specifically, it presents *ELAPSE*, a cooperative localization algorithm to support aerial mesh formation even in poor GPS conditions. Chapter 6 focuses on *Uhura*, a software framework that provides communication facilities for a swarm of UAVs by abstracting from the underlying M2M technologies. In addition, our tool supports automatic selection of the M2M stack on multi-radio ground/aerial nodes based on QoS requirements of the IoT application.

Chapter 2

State of Art

Nowadays, computer science is largely used in many different fields. For Structural Health Monitoring (SHM), as the main research field of this Ph.D., there are a lot of different aspects to investigate using a combination of multidisciplinary knowledge. In particular, this thesis describes how multiple heterogeneous networks can work together, like Ground Wireless Sensor Networks (WSN) and Aerial networks, in order to create an infrastructure with more capabilities useful for problematic IoT monitoring applications, e.g., SHM. To aim this, WSNs were largely investigated under many aspects; this chapter presents the state of the art of WSNs for Iot Monitoring 2.1.1. In particular, the SHM application was chosen. For the scope of this research, the networking aspect was mainly investigated; hence, the second part of this chapter is dedicated to WSNs in Mesh configuration 2.2 as it opens to new possibilities without special configurations, like more scalability and reliability of a WSN and, let to add mobile nodes like drones described in Section 6.5.

2.1 IoT Monitoring Systems

Advancements in technology and lower hardware costs have conducted to significant growth in the use of the Internet of Things (IoT) in the past decade [1]. IoT is applied in various domains, such as smart homes, agriculture, and grids, where multiple devices are connected to perform a specific task. Devices within the same system are connected through a network, like soil sensors monitoring humidity and temperature or cameras monitoring occupancy in a facility.

To ensure reliable communication between these devices, there is a growing demand for efficient and secure networking in IoT. This can be particularly challenging in rural areas, harsh environments, or devices with limited resources. To address these challenges, wireless sensor networks (WSNs) have become increasingly important [2].

WSNs consist of considerable low-power devices, such as sensors, that are capable of sensing, computing, and exchanging data. These devices are often used to collect data from the environment and transmit it to a central unit, known as a sink, for processing or storage. WSNs are crucial in various applications, including health, industrial, and environmental monitoring systems.

One of the most spread categories is the Urban domain. Here we can find some relevant sub-categories.

- **Smart Cities** WSNs in this kind of application can provide real-time data from traffic monitoring, smart car parking systems, or the current status of tunnels, bridges, and other public infrastructure.
- **Smart Home** WSNs can be used to monitor and control various devices and appliances in a home, such as lighting, heating, cooling, and security systems, providing convenience and energy efficiency.
- **Transportation Systems** this subcategory includes the monitoring and managing various modes of transportation, such as vehicles, trains, buses, and shipping containers. WSNs can be used to track the location and status of transportation assets, monitor fuel consumption, manage traffic flow, and optimize delivery routes, among other applications.

Another sub-category of the Urban domain is for sure Structural Health Monitoring. The next Section, as mentioned at the beginning of this chapter, is dedicated to WSNs for SHM applications, including detailed characteristics to understand better which kind of data these WSNs need to collect, elaborate, and communicate.

2.1.1 SHM using Wireless Sensors

The ability of structural health monitoring (SHM) employing wireless sensor networks (WSNs) to lower installation costs has attracted study attention.

As well as the upkeep of SHM systems, it is possible to extend the life of structures and increase public safety by using SHM systems, which have been utilized to monitor important infrastructure, including stadiums, highrise buildings, and bridges. WSNs for SHM have a high data-gathering rate, which presents special network design difficulties.

The process of structural health monitoring using WSNs requires the installation of a large number of sensors throughout a structure in order to collect data about the condition of the structure. This data is then processed to make decisions about the overall health of the structure. Historically, one of the first WSNs successful and largest installations was on the

Golden Gate Bridge in 2007 by a research team at the University of California in Berkeley [9]. In order to cover the length of the structure, they deployed 64 nodes reaching 46 hops from the start to the end, achieving a bitrate of 441B/s at the 46th hop. As another important deployment is the monitoring system of the Torre Aquila located in Trento(Italy) [10]. It is a 31-meter-high medieval tower that requires several sensors deployed so as not to disfigure it. Here are the advantages of WSNs: a civil engineer can move the sensors many times without expending money compared to the wired version and without ruining the structure.

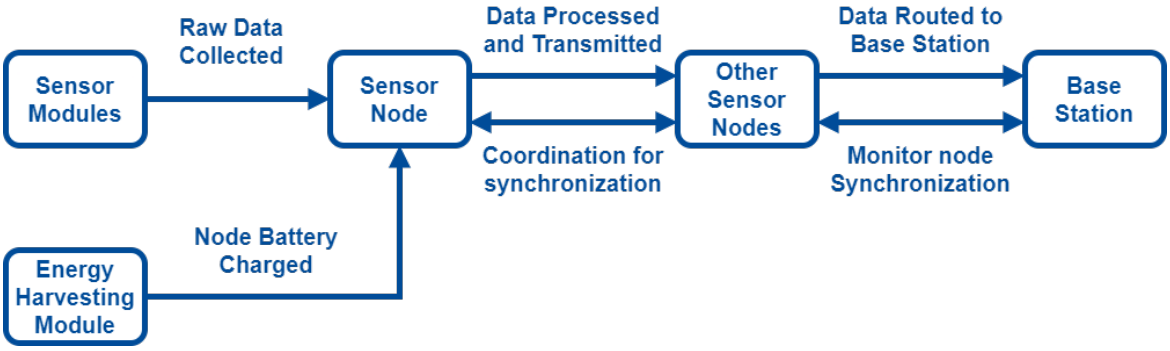


Fig. 2.1 process of structural health monitoring using wireless sensor networks

Sensor and Data for SHM

The approach for SHM using WSNs is shown in Figure 2.1. Everything starts from the module that sense and acquires data. In the next two subsections, I describe the data useful for SHM and which sensor can detect these parameters.

Collectable Data A SHM system typically collects data on the condition of a structure, such as cracks or changes in shape. This data can be used to assess the safety of the structure and to determine when repairs or maintenance are necessary. Typical data includes the temperature, humidity, light and sound. So far, the most commonly measured parameters for structures are their acceleration and velocity. From a design and analysis point of view, these data can be classified to explain easier which aspect can be investigated [11].

- **Load** The loads acting on a structure can be divided into two categories: environmental loads and loads due to passing vehicles. Environmental loads include factors such as wind speed and precipitation, while the weight of the vehicles themselves typically causes loads due to passing vehicles.

- **Global Load Response** The response of a structure to a given load that can be measured throughout the entire structure is called a global load response. The most common parameters that are measured are a structure's acceleration and velocity.
- **Local Load Response** The response of a structure to a local load can only be measured in a specific part of the structure. The measured parameters are typically strain, crack, and tension forces.
- **Environmental Factors** The environment surrounding a structure can significantly impact the structure's condition. Environmental factors such as temperature, humidity, and atmospheric acidity can contribute to environmental loads such as wind.

Sensor Types A specific sensor can collect the data described in section 2.1.1. As for the data, it exists different sensor types, and the most common are described as follows [11, 12]:

- **Accelerometers** An accelerometer for SHM consists of a mass attached to a spring. The mass is free to move in response to acceleration, and the spring restrains it. The accelerometer measures the acceleration of the mass in response to the applied acceleration. Also, a piezoelectric effect is used to detect the movement of the mass. Spring-mass accelerometers are typically more rugged and can withstand higher acceleration levels than piezoelectric accelerometers. They are also less sensitive to temperature changes. However, piezoelectric accelerometers are typically more accurate than spring-mass accelerometers [13].

Fig. 2.2 Accelerometer by Move Srl [14]

- **Strain sensors** Strain sensors used for SHM can be classified as piezoresistive or embedment strain gauge based. For the piezoresistive sensors, the changes in the

resistivity of a material caused by external strain is measured. While for embedment strain gauge based sensors, the external strain caused changes in the gauge resistance, which is then measured.

There are several types of strain sensors used for SHM, including optical fiber sensors, piezoelectric sensors, and piezoresistive sensors.

- Optical fiber sensors are based on the principle of Bragg grating, in which the external strain causes a change in the grating constant, and this change is measured [15].
- Piezoelectric sensors are based on the piezoelectric effect, in which the external strain causes a change in the electrical polarization of the material, and this change is measured [13].
- Piezoresistive sensors are based on the piezoresistive effect, in which the external strain causes a change in the resistivity of the material, and this change is measured [16].

- **Corrosion Sensors** Corrosion sensors used for SHM work on the principle of monitoring the corrosion of a metal by measuring the variable impedance of the metal at the frequency of the applied ac field [17, 18].

  The principle of the impedance sensor is based on the fact that the impedance of a metal is a function of the corrosion of the metal. The impedance of metal is affected by the presence of corrosion products on the metal surface. The impedance sensor measures the impedance of the metal at the frequency of the applied ac field. The sensor is able to detect the presence of corrosion products on the metal surface and monitor the corrosion of the metal.

Damage Detection and Localization

Once the sensor collects the data, the node can aggregate it or send it to the next node or directly to a sink node according to the measurement policy or the node's capabilities.

Damage Detection One of the main goals of structural health monitoring is to detect damage. This usually requires collecting sensor data that can be used to identify parameters related to the overall health of the structure. The most common parameters used for damage detection are modal parameters like natural frequency and mode shape. Modal parameter estimation can be done in both the time and frequency domains [19]. Once modal parameters

have been extracted, damage detection algorithms are used to determine if damage has occurred.

There are many different types of damage detection algorithms. Some of the most common include:

- **Change detection**: Compare modal parameters from a known healthy state to modal parameters from a suspected damaged state. If there is a significant difference, damage is likely present.
- **Correlation-based methods**: Use a correlation function to compare modal parameters from different states. If the correlation is low, damage is likely present.
- **Neural network-based methods**: Use a neural network to learn the relationship between modal parameters and damage. The neural network can then be used to predict damage based on new modal parameter data.

As mentioned above, damage detection algorithms can be classified into two distinct domains: time domain and frequency domain.

- **Time Domain Algorithms**

  The time domain algorithms are based on the analysis of the time history of the signals. The structural response is usually compared with the excitation that caused it. The excitation is usually known, and as a result, the comparison between the excitation and the response can provide information on the structure damage state.

  A common approach to time domain analysis is to use the impulse response function (IRF) of the structure [19]. The IRF is a linear operator that transforms a source into a response. Once the IRF is known, the structure's response to any arbitrary source can be found. When the structure is damaged, the IRF changes. The changes are usually small, but they are significant enough to be detected by the IRF.

  Time domain damage detection algorithms use the changes in the IRF to detect damage. The most common approach is to compare the structure's response to two different sources. The sources can be two different excitations, or they can be the same excitation at two different times. If the structure has not changed, the two responses should be identical. However, if the structure has changed, the two responses will be different. The difference between the two responses can be used to detect damage.

- **Frequency Domain Algorithms** Frequency domain algorithms are based on the analysis of the frequency content of the signal. The signal is usually the response

of the structure to an excitation. The excitation is usually known, and as a result, the comparison between the signal and the excitation can provide information on the structure damage state [19].

The most common approach to frequency domain analysis is to use the Fourier transform. The Fourier transform is a linear operator that transforms a signal into its frequency components. Once the Fourier transform is known, the frequency content of the signal can be found. When the structure is damaged, the Fourier transform changes. The changes are usually small, but they are significant enough to be detected by the transform [20].

Frequency domain damage detection algorithms use the changes in the Fourier transform to detect damage. The most common approach is to compare the signal to the excitation. If the structure has not changed, the two signals should be identical. However, if the structure has changed, the two signals will be different. The difference between the two signals can be used to detect damage [21].

Damage Localization After damage has been detected in a structure, it is necessary to determine its location. This process, called damage localization, requires the installation of enough sensors to provide coverage sufficient to locate damage anywhere in the structure. Insufficient sensor coverage can result in damage detection without localization. Commonly used damage localization techniques are frequency-based, mode shape–based, flexibility matrix-based, stiffness matrix-based, and support vector machine based.

- The main idea behind **frequency-based** damage localization is to compare the change in the fundamental frequencies of the structure with that of the damaged structure to localize the damage [22].
- The mode **shape–based** damage localization technique is based on the mode shape change of the structure [22].
- The flexibility **matrix–based** damage localization technique is based on the flexibility matrix, which is the inverse of the stiffness matrix [23].
- The stiffness **matrix–based** damage localization technique is based on the stiffness matrix [24].
- Support vector **machine–based** damage localization technique is based on support vector machines [25].

SHM WSN approach

As mentioned in the section above, there are two types of sensor networks that can be used for any sensing task. Especially for SHM, there are a lot of challenges to take into account before designing the topology. In this section, firstly, I describe the main advantages of a WSN compared to the wired version for SHM. Then, a full process of a WSN configuration is presented, starting from the collection data module to the monitoring one.

Sensor Networks The selection of the specific network configuration is not obvious; for sure, the WSN may be the easiest one for many good characteristics, but it's hard to keep reliable compared to the Wired. A Wired Sensor Network is formed by many nodes spread in the scenario, connected to each other by wire or directly to a base station that gathers the data collected by each sensor. The wire is not only used to transfer data, but also transfer the power source. So, by the nature of this approach, the deployment time is very long and expensive in terms of cost, and the number of sensors is limited since more space for the wire is needed. But as advantages, we have to consider the high quality of connection since the wire can reach high bandwidth, Data rate, and higher synchronicity. Latter is needed for the Damage Detection and Localization shown in section 2.1.1 and is crucial for the final data analysis.

On the opposite side, a WSN reduces the cost drastically, and the installation time is very short since we don't have to place a wire for every sensor module; also, it is possible to install many nodes. For these reasons is preferred for SHM applications due to the main requirements for Damage Detection and localization to have a minimum number of sensors spread on the structure. The disadvantages are basically two: the lifespan is related to the battery, so very limited in a scenario particularly challenging like SHM, and the connection capabilities are limited by the wireless technology used [26].

Challenges in WSN for SHM Compared to regular WSNs, WSNs designed for structural health monitoring have to tackle several distinct difficulties. These include the necessity of gathering, manipulating, and transmitting large amounts of data, the need for densely packed sensor nodes, and the long distance between nodes and base stations. We can find different WSN approaches to accomplish this kind of task in the literature. Starting with a general wireless network, it's possible to centralize the final analysis or let the sensor pre-process some data before sending the valuable one to the final destination. A fully centralized case described here [27] shows the time on collecting, compressing, and sending 1min of vibration of 6000 samples with a sampling rate of 100Hz. This system generates traffic of 667 Bps, and the authors say that it is not scalable due to network issues. So, the direction of the

research is to take into account the possibility of processing more data directly on the sensor as described by Hackmann [28] in order to reduce the payload to send. In this case, with a sampling rate of 560Hz, the authors decide to perform the process phase with a reduced sample array, collecting processing and sending in 5s, 3.7s of vibration, and obtaining 2048 samples. Of this final time, 1.3s are used to apply Fast Fourier Transform (FFT) algorithm for curve fitting, reducing the payload to 300Bps. Even if they pre-process data directly on the sensor, they don't take into account the possibility of sharing useful data between the sensors and the sink node, and this is what Dos Santos made in his work [29]: 512 samples at 1000Hz rate for 0.5s vibrations time. The system takes 9.5s to perform FFT, peak extraction, data collecting by the sink node, coefficient calculation, and collaborative decision. This processing time reduces drastically, the payload to 28Bps. Of course, the trade-off of this kind of system is related to the power calculation of the sensor itself, so in the end, a balance between network capabilities and the sensors (including energy consumption) is needed for SHM applications using WSNs.

2.2 Wireless Mesh Sensor Networks

As shown in the previous section, some WSN applications may produce large amounts of data and needs sometimes a short response, which can be a heavy load on networks. Mesh networks can be used in situations where the structure or shape of the network does not allow each node to be within the range of its final destination, improving the scalability and reliability of the network. Routing protocols are a crucial component of Wireless Mesh networks (WMNs) [30] as they determine the best path for data to travel from one node to another in the network. The choice of a routing protocol can have a significant impact on the performance of the WMN, including factors such as network scalability, reliability, and power consumption. In recent years, the IT industry has dramatically increased the use of wireless devices and mobile networks. This expansion was accompanied by developing of various routing protocols responsible for delivering data packets across the network. They optimize the use of network resources, such as power consumption, processor time, memory, etc., allowing for maximizing the network's lifetime. In this section, I will describe some advantages of using WMN topology in conjunction with WSN [31], starting to describe WMNs, then moving on to introducing some possible cases study for Wireless Mesh Sensor Networks (WMSN). I will end by citing some related works about WMSN.

Fig. 2.3 Wireless Mesh Sensor Network concept

2.2.1 Wireless Mesh Networks

A Wireless Mesh Network (WMN) is a type of wireless network infrastructure that is composed of radio nodes interconnected by wireless links. The main characteristic of WMNs is their ability to provide strong, secure, and reliable wireless connections between multiple nodes, enabling communication over a wide area. The nodes in the network act as routers, relaying data from one node to another, thereby allowing data to travel over multiple hops [32]. This feature enables WMNs to offer robust and flexible wireless connectivity solutions, making them an attractive option for various applications, such as disaster response, rural connectivity, and smart cities.

WMNs are used in various applications, from providing wireless access for urban areas to providing secure wireless connectivity for enterprise networks. They are also increasingly being used in disaster relief scenarios to provide communications networks for emergency responders. From a technical point of view, WMNs are attractive for many reasons. WMNs are typically more reliable than other wireless networks, like star topology, due to their mesh topology and redundancy, allowing for communication even if some nodes are lost or fail. Additionally, WMNs are scalable and can easily be expanded to cover a large area, making them ideal for large-scale applications. Finally, WMNs offer lower latency than other wireless networks and can typically support higher data rates.

Some technical key features of WMNs are described deeply as follows [33]:

- **Self-healing capability**: WMNs employ a self-healing mechanism that utilizes redundant paths and dynamic routing protocols to automatically reroute data packets in case

of node failure or disconnection, thus ensuring that the network remains operational and minimizing disruption to users.

- **Scalability**: WMNs are designed to easily expand and accommodate new devices, as well as increase capacity by adding new nodes to the network. This is made possible by using mesh topology, which allows for seamless integration of new nodes without the need for reconfiguration.
- **Redundancy**: WMNs utilize multiple paths for data transmission by using routing protocols that explore multiple routes between source and destination. This increases the reliability and availability of the network and reduces the risk of network failure.
- **Multi-hop routing**: WMNs employ multi-hop routing protocols, which allow data packets to travel through multiple nodes before reaching their final destination, allowing for efficient use of network resources. This improves overall network performance, as well as reduces congestion.
- **Support for multiple wireless standards**: WMNs can support various wireless standards, including IEEE 802.11 (Wi-Fi), IEEE 802.15 (Zigbee), and IEEE 802.16 (WiMAX), making them versatile and flexible. This allows for a wide range of devices and communication protocols to be accommodated by the network.
- **Quality of Service (QoS)**: WMNs are designed to support Quality of Service (QoS) mechanisms that allow for prioritization and allocation of resources to different types of traffic, such as real-time video or voice, to ensure that critical applications receive the necessary bandwidth.
- **Security**: WMNs employ a set of security measures, such as encryption and authentication, to protect against unauthorized access and data breaches. These measures include the use of encryption to protect data as it travels over the network and authentication to ensure that only authorized devices can access the network.

Routing Protocols

In WMNs, several types of routing protocols can efficiently route data between nodes efficiently [34].

Reactive routing protocols, such as *Ad-hoc On-demand Distance Vector* (AODV) and Dynamic Source Routing (DSR), are used to establish routes on an as-needed basis. These protocols work by flooding the network with route requests when a node needs to send data to a destination it does not have a route. A route is established only when a node that has

a route to the destination responds to the request. These protocols are highly adaptive to changes in the network topology, but they can introduce delays in route establishment and may generate high control overhead in highly dynamic networks. AODV is a reactive routing protocol that is used to find routes between nodes in an ad hoc network. It uses a combination of both distance vector and link state routing techniques. DSR is another reactive routing protocol which is used to find routes in wireless ad-hoc networks. DSR uses source routing, which means that the sender of a packet specifies the complete route that a packet should follow through the network.

Proactive routing protocols, such as Destination-Sequenced Distance Vector (DSDV) and Optimized Link State Routing (OLSR), maintain a table of all destinations in the network by periodically distributing routing table updates to all nodes. These protocols are able to route data between nodes as routes are already established quickly, but they can generate high control overhead and may not adapt well to changes in the network topology. DSDV is a proactive routing protocol that is used to find routes between nodes in an ad-hoc network. DSDV uses a distance vector routing technique and is based on Bellman-Ford algorithm. OLSR is another proactive routing protocol that is used to find routes in wireless ad-hoc networks. It uses a link state routing technique and is based on the shortest path first algorithm.

Hybrid routing protocols, such as Zone Routing Protocol (ZRP), combine the best of both worlds by using a proactive scheme for local routes and a reactive scheme for remote routes. This can reduce the control overhead of route discovery and allow for fast route establishment. ZRP is a hybrid routing protocol that is used to find routes in wireless ad-hoc networks. It uses a combination of both proactive and reactive routing techniques. ZRP divides the network into smaller zones and uses proactive routing within a zone and reactive routing between zones.

It is important to note that different routing protocols have different trade-offs, and the choice of which protocol to use depends on the specific requirements and constraints of the wireless mesh network. Factors such as network size, mobility, and traffic patterns should be taken into consideration when selecting a routing protocol for a wireless mesh network.

Routing metrics

In wireless mesh networks (WMNs), routing metrics are used to determine the best path for data to travel from a source node to a destination [35]. The most commonly used metric is the hop count, which selects the shortest path based on the number of nodes between the source and destination. However, this metric has been recognized as a limitation in WMNs as it can result in congested paths. To address this limitation, researchers have begun using

quality-aware metrics, which take into account factors such as link loss ratio and transmission rate. One example of this is the expected transmission count (ETX) metric, which estimates the number of transmissions required for successful data delivery. Nonetheless, it does not consider bandwidth, packet size, or link interference. Another metric, expected transmission time (ETT), considers packet size and link bandwidth but not load or link interference. An alternative, interference and channel switching (MIC), is topology-dependent and selects paths with the least number of nodes that share the wireless channel; however, it does not indicate if the interfering node has data to transmit. Another factor that is taken into account is the transmission rate, there are several widely adopted algorithms that help in controlling the transmission rate, such as automatic rate fallback (ARF) or adaptive ARF, and recently developed rate adaptation algorithm based on reinforcement learning (RARE). The choice of which metric to use depends on the specific requirements and constraints of the wireless mesh network.

2.2.2 Usage of WMN with WSN

In section 2.2 I introduced how WMNs can enhance WSNs. At this point, using a fully connected network opens new possible use cases. Overall, WMNs are an increasingly popular technology that can be used for a variety of purposes, such as monitoring and control in industrial and transportation networks, monitoring and security in extreme environments, and surveillance in public spaces. Furthermore, WMN can provide reliable communications, accurate data collection, and real-time control systems, making it a powerful tool for many applications[3].

- **Smart Home**
  The implementation of Internet of Things in the home environment brings with it a variety of benefits. Smart lights, TVs, and robotic vacuum cleaners are some of the products already available on the market. However, making a home 'smart' requires more than just connecting an existing device to Wi-Fi. Wireless sensors and controllers must be placed in the room, on pipes, and even on the floors and walls to collect and transmit data. This data should be primarily processed at home, as this reduces the amount of data traveling across the external network and helps protect privacy. In this situation, a WMSN topology is ideal for creating a smart home environment, as it allows for the installation of a gateway with a dedicated operating system (OS). This OS can be used to manage better and deliver the data, whilst also allowing for local processing, reducing the strain on the bandwidth. The use of WMSN also supports the development of home automation applications. These can be used to control

lighting, heating and ventilation, as well as provide the ability to monitor and control appliances and other devices in the home. WMSN can also be used to create an efficient and secure home network, allowing multiple devices to connect to the network with minimal interference and latency[36, 37].

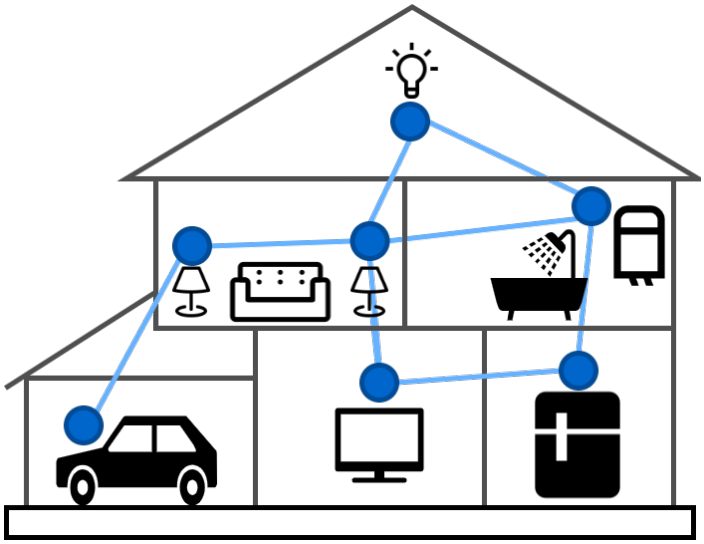


Fig. 2.4 Smart Home

- **Smart City**
  WMSN can be implemented in smart cities to create a Smart Vehicle Infrastructure that can detect and prevent road accidents. The WMSN will consist of anisotropic magnetoresistance circuits for vehicle detection, radio modules, rechargeable lithium-ion batteries, and a WMSN coordinator. When a vehicle passes over a wireless node, the sensor device records the detection, marks the time stamp, and sends the record to the WMSN coordinator. This information data can be transmitted over long distances with less delay, creating a large area of coverage. The Smart Vehicle Infrastructure will also include a Vehicle Adhoc Network (VANET). This network will have randomly organized nodes that are mobile, with travel speeds of up to 120 km/h. The Roadside Infrastructure Elements (RIE) of the network will be stationary and connected to other objects in the wireless sensor mesh network. When a road accident is detected, the car stop action is initiated for the approaching vehicle, and a message is sent to the RIE to prompt the departure of emergency vehicles[3, 38].

- **Large-Area Monitoring**
  WMN has been gaining traction in recent years for its superior features such as high-speed deployment, flexible structure, high bandwidth, NLOS transmission and

reliability, and their ability to connect to the Internet, Wi-Fi LAN, PSTN, and other networks. This technology has many advantages in the industry of electric power, healthcare, construction, logistics, Industry 4.0, and more. In order to implement control and transfer of sensor information in a large industrial facility, a wireless mesh sensor network combining sensing, distributed information processing, integrated computing and reliable communication technologies are created. This network allows for accurate data collection to ensure precision and high status in personalized or small-sized products. The goal is to coordinate the perception, collection, and processing of information about perceived objects in the network coverage area. This information can be used to track and monitor the status of equipment and the environment and to create real-time control systems. WMN can also be used for the purpose of monitoring and control in transportation networks. Here, WMN can be used to optimize the efficiency of transport and reduce the cost of operations by integrating the transportation network. Additionally, WMN can be used for the purpose of monitoring and security in extreme environments, such as the hazardous environment. Here, WMN can be used to provide reliable communication and monitoring of personnel and equipment. Finally, WMN can be used for the purpose of surveillance in public spaces, such as airports and stadiums. Here, WMN can be used to monitor areas of interest, detect suspicious activities and take appropriate action[39].

The introduction of mesh capability to traditional WSNs has a significant effect on the commercial expansion of this technology and provides new chances for applications or markets such as situational awareness and locating assets, safeguarding firefighting activities, streaming media services, or monitoring environmental security. In order to accomplish this interconnection objective, certain essential needs must be taken into account during the design phase of a WMSN, which impact the performance of a WMSN; these can be outlined as follows.

- **Energy Management**
  The energy management policy and energy efficiency design is essential for the design of WMSN networks, and it is a significant issue. Typically, power for the devices is supplied by AA/AAA batteries, which reduces the lifetime of the elements during continuous operations. In some cases, solar or wind energy sources are too costly or impractical to use. This is especially true in large-scale mesh networks, where the number of hops between source and destination can be very large, and thus the power needed for data transmission/retransmission may reduce the network's lifetime. Power management mechanisms can address these problems by decreasing power consumption by turning off certain components of a node, such as a processor or radio

module. Other considerations, such as addressing schemes, routing procedures, or security, should also be taken into account to prevent overloading the nodes, which will increase the energy requirement.

- **Link Reliability**
  Ensuring that there is a reliable connection between the source and destination nodes throughout the application's operation is a vital factor in the design of WMSN networks. The mesh topology approach can provide dependable connectivity as it offers multiple paths to link source-destination pairs. This allows for alternative paths to be used in the event of, for instance, a dead intermediary node. Additionally, the need for reliable communication can be addressed by increasing the number of nodes in the restricted area, thus leading to more redundancy in the path. This offers more reliable connectivity to all of the nodes in the particular area as they would have more neighbors in the coverage region.

- **Robustiness**
  WMSNs must possess the capability of being flexible, robust, and self-healing in order to effectively address issues such as changes in the network topology due to sudden node appearance and disappearance or to manage hostile and dynamic environment conditions or radio interference. Radio interference is often caused by external radio waves from other wireless technologies, such as Wi-Fi devices, which use the same transmission medium as WMSNs and also operate within the same frequency range. Therefore, in order to guarantee the redundancy of network routes and the implementation of effective mechanisms for mitigating the effects of external interference, WMSNs need to be able to maintain these properties.

- **Scalability**
  The requirement of scalability is becoming increasingly crucial to guarantee reliable communication, robustness, and efficient power consumption as the number of devices increases and the traditional WSNs expand to include hundreds of battery-powered network elements. Therefore, it is important to ensure that network performance does not degrade with the expansion of the WMSN

- **Interoperability**
  To ensure compatibility and interoperability across heterogeneous networks, including various wireless/wired technologies, WMSN protocols must be able to interoperate multiple networks without significantly increasing the complexity of software and hardware components.

- **Self-Organization**
  The most attractive aspect of a WMSN network for end-users is the minimal or no effort required to maintain it. However, for developers, self-organization is not a simple task. In order for these nodes to autonomously form a mesh network topology, they must be equipped with efficient hardware and software modules, which may result in a significant trade-off in terms of energy efficiency.

- **End-To-End Reliability**
  Ensuring that data is correctly delivered to its intended destination is essential. As stated before, the reliability of communication channels can be increased by adding more nodes to the service area. This will help reduce the rate of lost messages in the WMSN that can be caused by fading due to a broken connection in a wireless environment. Additionally, it is important to note that there are applications, such as personal health monitoring or the exact location of assets, that require transport-level services or notifications of significant events, such as the detection of intruders in secure areas. Thus, utilizing methods that prioritize data can also enhance the reliability of the overall network.

- **Mobility Support**
  In certain applications, such as large networks, it may be beneficial to ensure transmission latency and ensure data delivery from sources to recipients by moving receivers to different strategic points, even though many WSN or WMSN solutions are usually designed for static devices that remain in their original destinations.

2.2.3 WMN Technologies

Since one of the main aspects of this thesis involves Mesh networks as network topology, it is important to spend more words on the possible technologies able to do that. In the market, there are a lot of them specifically created for a task to accomplish the requirements discussed in 2.2.2. It is sometimes necessary to extend a standard in order to create the mesh system itself or maybe create a proprietary one from scratch. In this section, I want to cite ZigBee, DigiMesh, and Bluetooth Mesh as the most interesting technologies for WSNs used in this thesis.

ZigBee

Zigbee, named after the zigzagging pattern of honey bees between flowers, is a widely used Wireless Sensor Network standard with low power, low data rate, low cost, and short time

delay properties, making it easy to develop and deploy. It provides strong security and reliable data[40]. The ZigBee Alliance, consisting of hundreds of members such as Ember, Freescale, Chipcon, Invensys, Mitsubishi, CompXs, AMI Semiconductors, and ENQ Semiconductors, is the developer of the ZigBee protocol. ZigBee and IEEE 802.15.4 are distinct; ZigBee is a proprietary network protocol created by the ZigBee Alliance, which utilizes the transport services of the IEEE 802.15.4 network specification. It is analogous to TCP/IP using IEEE 802.11b network specification. The ZigBee Alliance is responsible for defining the network, security, and application layers, while IEEE 802.15.4 is responsible for the physical and media access control layers of LR-WPAN. It can reach up to 150 meters outdoors thanks to Direct Sequence Spread Spectrum (DSSS) technique consuming less power than Frequency Hopping Spread Spectrum (FHSS). ZigBee frequency bands are 868MHz (Europe), 915MHz (North America and Australia), and 2.4Ghz (worldwide), reaching up to 20kbps, 40kbps, and 250kbps data rates, respectively. As logical roles in a ZigBee network, we need a Coordinator, Routers, and of course, End nodes:

- A Coordinator is the network's root and can be the bridge of multiple networks. Takes care to initialize the network and choose the radio frequency channel, network identifier, and other operating parameters.
- Routers are the intermediate nodes of a ZigBee network. They can relay data from one node to the others.
- End devices are the most basic device in a ZigBee network. They can communicate only with Routers or the Coordinator, and since they are not used frequently, they can go into sleep mode by design.

All the possible topologies using the roles described above with AODV as the routing algorithm are:

- Star Topology: One coordinator and many End Devices connected to it directly. In this case, ZigBee uses a master-slave model
- Cluster Tree Topology: sharing some aspects with the Star version, but in this case, some of them can create a chain formation in order to expand the size of the network.
- Mesh topology: Each node can communicate with others if they are routers or the coordinator and in wireless range space. End devices are connected to them as they are the outline devices of the network.

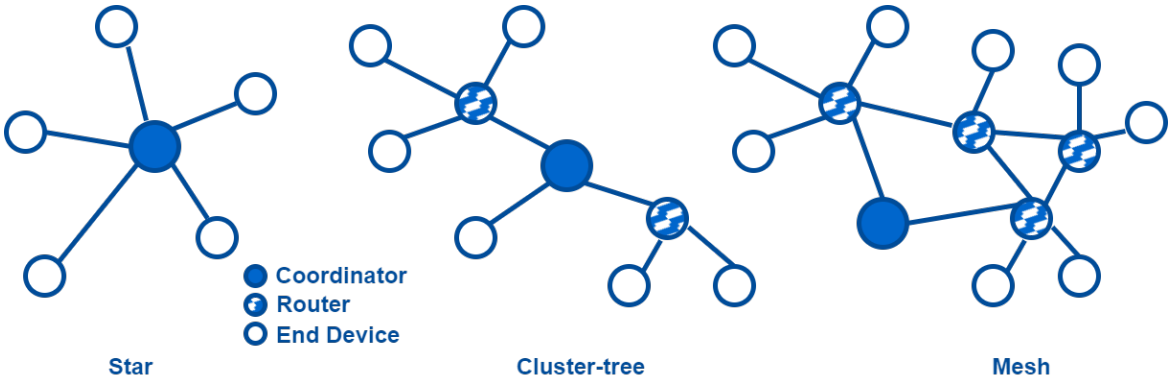


Fig. 2.5 ZigBee Topologies

XBee DigiMesh

XBee DigiMesh is a proprietary protocol made by Digi International over the IEEE 802.15.4 network specification as the ZigBee. Works on 900Mhz and 2.4Ghz frequency bands using FHSS and DSSS, respectively [41].

This protocol is used by all of the DigiMesh devices and implements characteristics as follows:

- **self-healing** system since any node can join or leave the network at any moment
- **all nodes are equals** so there is no father-son relationships
- **Silent protocol** for routing heading, a custom extension of Ad hoc On-Demand Vector Routing (AODV)
- **Route discovery** without keeping a route map but discovered when they are needed
- **Selective ACKs** for data transmission reliability
- **Sleep Modes**: for low energy consumption modes using synchronization mechanisms to wake up simultaneously.

By default, as all the nodes assume the same role, it creates a mesh topology. It is also possible to create a Star topology activating the "Waspmote" on each edge node: basically, force the node to communicate through only one Xbee Device (gateway in this case).

As mentioned above, the routing algorithm uses a reactive method derived from AODV in conjunction with an associative routing table to map the destination node's addresses for the next hop. A Route Discovery Process is used when a route fails or doesn't exist (thanks to the ack system). This process broadcasts a Route Request (RREQ), and any node

that is not the final destination receives this message is called an intermediate node. The intermediate nodes can drop the RREQ or forward if it has better route information, updating the info on the request and then broadcasting again. Once the destination node receives an RREQ, it unicasts a route reply(RREP) back to the source node using the path saved on the RREQ. This can happen many times so that the source can select the best route based on the round-trip quality.

Bluetooth Mesh

Bluetooth Mesh is a software standard built on Bluetooth Low Energy (BLE). The first version was released in 2017, enabling a mesh mechanism without changing the hardware 2.6. This opens the possibility of connecting every device already equipped with at least a Bluetooth 5.0 Module in a mesh topology, like smartphones, headsets, wearables, smart cams, etc. Based on BLE, it uses 2.4GHz and different payload sizes depending on the network topology in use: for mesh, it is 29 bytes [8, 42].

Internally, a node can assume different roles depending on the complexity of the tasks and can possess any of the following four features:

- **Low-Power Feature** Nodes that are limited in terms of power can utilize the low-power feature to minimize the time their radio is on, thus saving power. Low-power nodes (LPNs) work together with their companion nodes.
- **Friend Feature** Nodes with no limitations on the power supply are ideal for functioning as friend nodes. Such friend nodes are responsible for preserving and guarding messages and security updates meant for LPNs; when requested by an LPN, the friend node can provide the required data.
- **Relay Feature** Relay nodes act as transmitters, taking in messages and then sending them out to other nodes, thus allowing for a larger communication network. The capabilities of a node in terms of power and computing ability will determine how well it can perform this function.
- **proxy Feature** Proxy nodes facilitate the transmission and receipt of mesh messages between GATT and Bluetooth mesh nodes. This role necessitates a reliable power source and adequate computing power.

Element A single node can handle independent tasks like turning on light and reading temperature from a sensor called Elements. At least one default element must be implemented

on a Bluetooth mesh node, and many elements are possible. Each element has a unique address, enabling unicast on each of them if needed.

Models and States A node's functionality and behavior are determined and put into practice by models. Models exist within elements, and elements must include at least one model. Models define and carry out the functionality and behavior of a node, and states outline the condition of elements. This is applicable regardless of whether the nodes are connected on a factory floor, hotel, office building, or business campus.

Bound states refer to when a change in one state affects another state. A common example is the connection between level states and On/Off states. For instance, when the level changes from 0 to 1, the On/Off state automatically transitions from off to on. Bluetooth SIG-adopted models are identified by 16-bit codes, while vendor models are identified by 32-bit codes (16-bit Bluetooth-assigned company identifier and a 16-bit vendor-assigned model identifier). This ensures each model is uniquely identifiable. Bluetooth mesh networks communicate via a client-server architecture, where the server's purpose is to expose the states of an element. A basic state is a binary switch, which is either on or off. A Generic On/Off Server Model contains the state of the switch, and a Generic On/Off Client Model is used to control the Server Model by sending messages. This allows the client to turn the light on or off.

Model Types In the end, the client-server architecture defines 3 types of models:

- Server Model defines the element's behavior along specific messages
- Client Model defines the messages used as requests to change / read states of a server
- Control Model is a combination of Server Models / Client Models on the same node plus control logic to coordinate interactions between them.

Addresses As mentioned above, each element has its own address. In a Bluetooth mesh network, there are four types of addresses, and three of them are used for messaging:

- **Unassigned Address** Elements that have not been set up with a specific address have an unassigned address, meaning they cannot be used for messaging purposes since they do not have a distinctive address.
- **Unicast Address** When provisioning, a provisioner assigns a unique address to every element in a node that will be used throughout the node's lifetime on the network. Unicast addresses may be seen in the source and/or destination address of a message. Messages sent to a unicast address are only received and acted upon by one element.

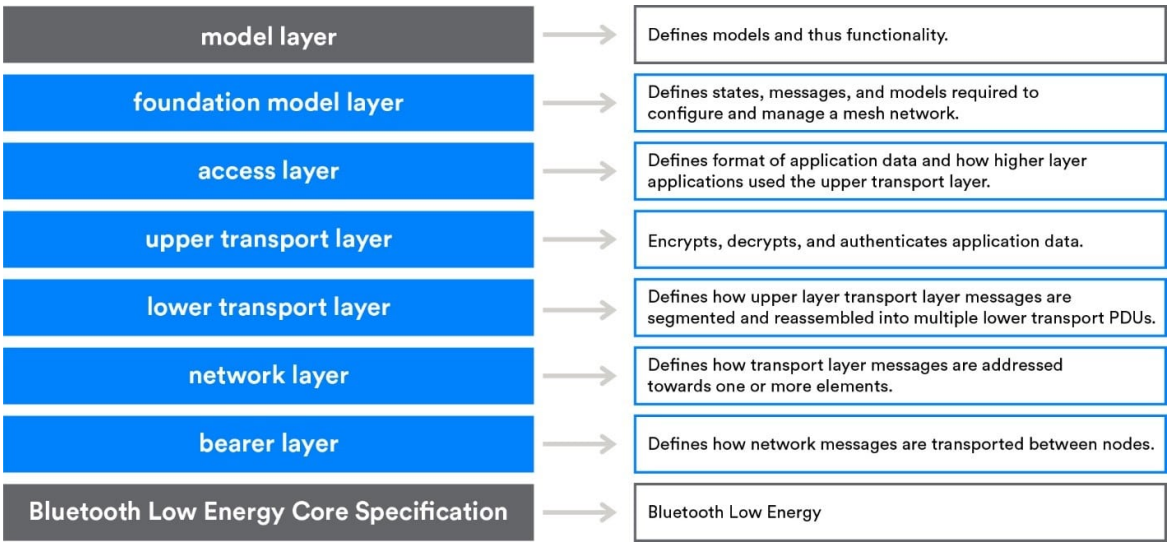


Fig. 2.6 BLE Mesh Architecture.[43]

- **Virtual Address** Virtual addresses are associated with a unique 128-bit Label UUID, which can be used to publish or subscribe to various elements. These elements can originate from a single node or multiple nodes.
- **Group Address** Group addresses are another type of multicast address found in Bluetooth mesh networking. There are two types of group addresses representing multiple elements from one or more nodes: static and dynamic. Static group addresses are pre-defined and cannot be changed, while dynamic group addresses are assigned on the fly by the network and can change. Group addresses can be used to send messages to multiple nodes or elements at the same time, like "all proxy nodes" or "all friends nodes".

2.3 UAV-aided sensor networks

UAV-aided systems are systems that utilize unmanned aerial vehicles (UAVs) to assist in the completion of tasks. UAVs can be used to gather data on the environment, such as images or video, as well as to provide coverage for areas that are difficult to access by traditional ground vehicles. Additionally, UAVs can also be used for surveillance and tracking, providing an extra layer of security. UAVs can also be used to deliver packages or to provide remote medical assistance, making them versatile technology for various fields.

UAV-aided wireless sensor networks are a new technology that combines the advantages of both UAVs and wireless sensor networks. UAVs can extend the range of a wireless sensor

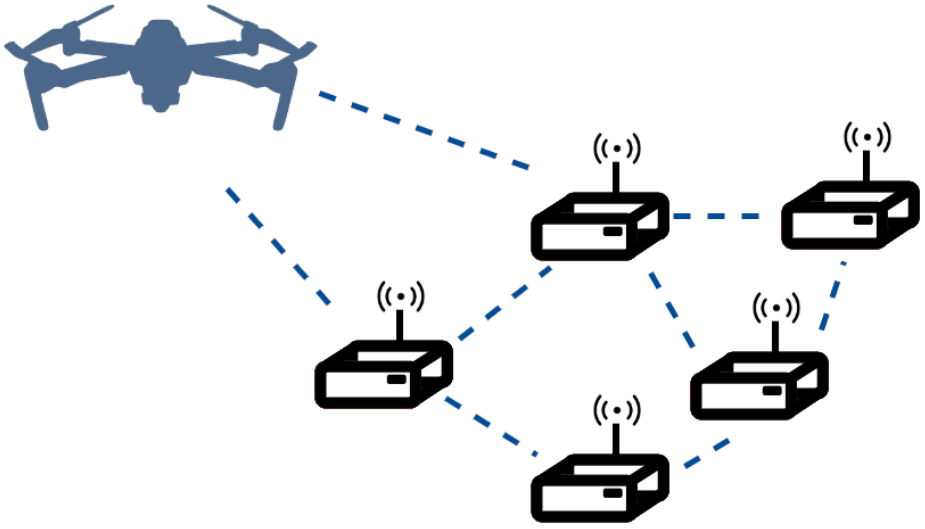


Fig. 2.7 UAV-aided Wireless Mesh Sensor Network

network, enabling sensors to be placed in areas that would otherwise be difficult or impossible to access. UAVs can also be used to increase the speed of data transmission, enabling data to be collected more quickly. They can also be used to monitor a wide area, allowing for more comprehensive monitoring than is possible with a single wireless sensor. In addition, UAVs can provide additional power to wireless sensor nodes, allowing them to operate for longer periods of time and in more remote locations. Finally, UAVs can provide a secure platform for data exchange, ensuring that data is not intercepted or tampered with.

In the literature, is possible to find theoretical works such as protocol simulations and path planning over the WSNs, as presented in the following, but lack documented real use cases regarding drone-sensor communication, an important aspect of this research work.

2.3.1 UAV-Sensor protocols

Talking about UAV-aided WSN, an important aspect to consider is certainly the technology behind the UAV-Sensor communication. This kind of communication can be managed by using well-known wireless technologies such as WiFi, ZigBee [44], and LoRa [45], but they are not designed to be power efficient enough for this kind of application. Since at least the drone involved in the communication has a limited lifespan caused by the battery, a different approach is needed for this kind of transmission. Researchers are investigating ways to reduce the energy consumption of the media access control (MAC) layer of the

communication stack. This includes optimizing the transmission power and data rate, as well as investigating energy-efficient channel access and data scheduling algorithms. Regarding the MAC protocols, they are largely surveyed by Poudel et al. [46] and explain in detail which ones are designed for specific QoS. For example, `PESC`[47], `CSMA/CA-based cooperative relay and heuristic algorithm` [48], and `AD-PS MAC`[49] focus on the energy aspect; `PCWAS` [50] and HP-MAC [51] focus on low-delay transmissions.

2.3.2 Energy Management

Many works regarding UAV-aided power management for WSN are in the literature. It covers the possibility of using UAVs to recharge each sensor using wireless energy harvesting [52] or improving the communication aspect [53], using them as Data Mules with optimal path planning considering the environment. In [54], the authors studied how hilly or mountainous areas are difficult to build and maintain for infrastructure-less WSNs, finding optimal altitude transmission for each cluster deployed on the scenario to avoid lossy communication and then extra power consumption. Since the UAVs have limited energy, each sensor can assist it in preparing useful data before the collecting phase. Bin Liu *et al.* [55] investigates the possibility of changing the transmission mode between waiting for the UAV, conventional sink node transmission, or uploading data to UAV when possible within a time tolerance. This kind of approach aims to guarantee data delivery and, at the same time, save energy on both the sensor node and UAVs.

2.3.3 Data Collection

The integration of unmanned aerial vehicles (UAVs) with wireless sensor networks (WSNs) has the potential to improve data collection and overall network performance significantly. UAVs bring a unique advantage to WSNs by enabling data collection from various locations within the network, providing a more comprehensive and accurate picture of the monitored environment.

Haider *et al.* [56] demonstrated how UAVs can improve WSN data collection through effective path planning. By dividing the environment into cells and creating multiple clusters of nodes, the UAV can efficiently collect data from each cluster, allowing the nodes to focus on their primary tasks without being burdened by communication-related issues.

Juan Liu *et al.* [57] tackled the age-optimal data collection problem for UAV-aided WSNs. They proposed two optimization problems to minimize the Age of Information (AoI) at each sensor node, which is calculated based on the data uploading time and the UAV's flight time after leaving the sensor area. Through simulations, they designed and evaluated a

strategy to achieve better AoI performance, which is crucial for real-time monitoring and decision-making applications.

Chixiong Mao *et al.* [58] expanded on the work of Juan Liu *et al.* [57] by introducing the use of multiple UAVs to address the same problem. They considered power consumption as an additional factor and proposed a solution that minimizes the AoI while reducing the energy consumption of the network. The use of multiple UAVs provides even more opportunities for improving data collection and network performance.

Overall, the integration of UAVs with WSNs brings new challenges and opportunities for research and is expected to continue evolving to meet the demands of various monitoring and control applications.

Chapter 3

Objectives and Methodologies

The chapter introduces scenarios and macro-objectives of the thesis, which are investigated in detail in the following chapters. First, in Section 3.1, the definition and components of Hybrid Wireless Mesh Sensor Networks (H-WMSN) are provided. Scenario assumptions and requirements are briefly discussed in Section 3.2. Finally, research objectives and methodologies are presented in Sections 3.3 and 3.4, respectively.

3.1 Scenario

In Section 2, we introduced UAV-aided WSNs as special instance of hybrid aerial-ground networks, in which the mobile nodes can support effective and energy-efficient data collection in harsh environments. This is the case with IoT monitoring systems which must be deployed in rural areas with limited wireless coverage and no access to stable/wired power suppliers.

Although the thesis addresses general research problems of IoT monitoring systems, in the following, we refer to a specific application related to Structural Health Monitoring (SHM) of critical infrastructures (e.g., a bridge) only as an example of a target scenario that meets requirements and assumptions of our work. Figure 3.1 shows the overall scenario. We assume that multiple sensor nodes (denoted as red squares in the Figure) have been deployed at specific locations of the target structure, in this case, a bridge. Without loss of generality, we also assume that the WSN nodes will sense the environment and offload their data to a Sink Node (denoted as blue circles in Figure), which works as a network data aggregator. Multiple paths may be available between a Sensor and a Sink node, hence creating a ground Wireless Mesh Sensor Network (WMSN). As pointed out in Section 2, several Machine to Machine (M2M) technologies can be used to build such topology. At the same time, the Sink Node may not be equipped with reliable or stable connections to a remote unit. For this reason, Unmanned Aerial Nodes (UAVs) can be deployed as data mules, i.e., hovering

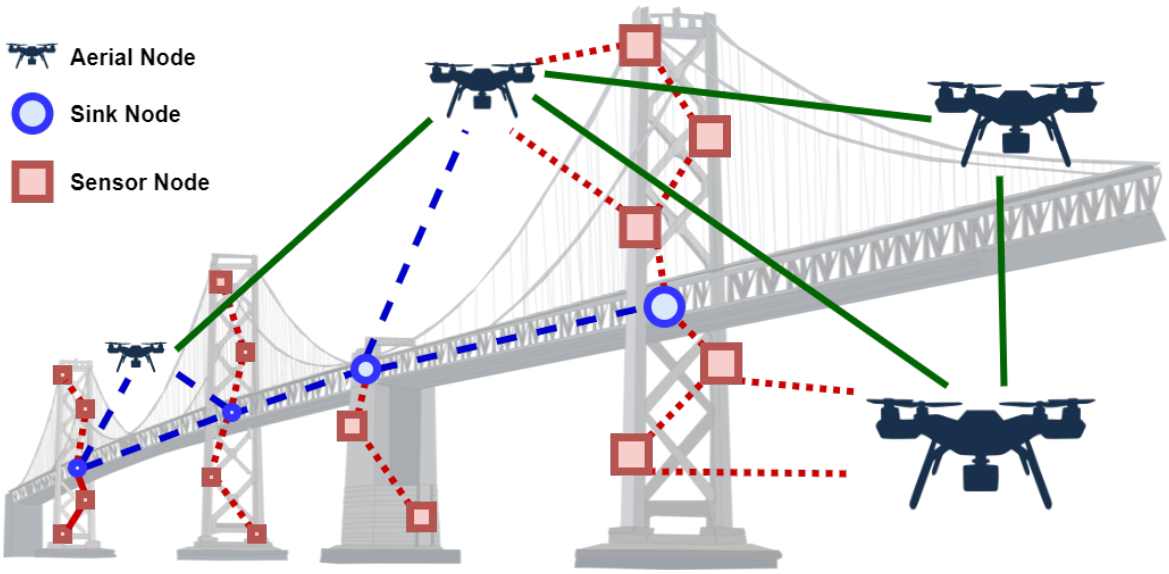


Fig. 3.1 Scenario investigated in this thesis.

over the Sink Node and collecting data from the environment. In the presence of multiple UAVs, the latter can create a swarm and build an aerial WMSN, depicted through green lines in Figure 3.1. In this thesis, we focus on the possibility of integrating the aerial and ground WMSNs through specific UAV-to-Sink links. We refer to such integrated network infrastructure as **Hybrid WMSNs (H-WMSN)** in the rest of the thesis. In the following, we detail the characteristics of nodes that build the H-WMSN:

- **Sensor Node.** This represents a computational unit able to sense the environment, process the sensing data, and communicate with the Sink and other peers via short-range M2M technologies. In the specific SHM use case, the sensor nodes could measure the overall vibration of the structure through transducers such as accelerometers, strain gauges, and inclinometers. The type of sensing data is out of scope of the thesis, which is application-agnostic. Sensor nodes can also serve as relays in order to forward the traffic from other peers and establish multi-hop connectivity within the ground WMSN.

- **Sink Node.** Sensor nodes are clustered, as depicted in Figure 3.1. Each cluster has a Sink Node collecting the data from sensor nodes of relevance. We assume each Sink node to be provided with one or multiple M2M communication technologies in order to extend the capabilities of the data offloading process. The supported M2M communication technologies can be short or long-range, red and blue lines in Figure, and each of them may support different Quality of Service (QoS) characteristics in terms of throughput, packet delivery rate, and latency.

- **Aerial Nodes** with autonomous mobility can fly over the scenarios at periodic intervals and hover over the Sink Nodes. Based on the requirements of the IoT monitoring application, they can be provided with sensing capabilities (e.g., camera). As depicted in the Figure, multiple UAVs may be deployed, especially on large-scale scenarios, and create a fully connected aerial WMSN. In addition, each UAV may be equipped with multiple M2M technologies to communicate with ground devices (i.e., Sink nodes) and with other peers. In this thesis, we investigate two possible use cases of aerial nodes. In the first case (data muling), the UAVs serve as mobile data collectors; hence they must be able to establish a reliable communication link with the ground WMSN and gather the sensing data from the Sink Node. In the second case (self-healing WSN), the UAVs must be able to temporarily replace the operations of faulty ground devices, being a Sensor and Sink Node, and support the data acquisition in such extreme situations. The two use cases can be considered complementary (i.e., one UAV acts as a data mule while the other replaces the operations of a faulty sensor/sink node).

3.2 Requirements and Challenges

Generically speaking, the deployment of H-WMSNs to support IoT monitoring systems is highly challenging and should take into account a wide range of requirements, which depend on the specific application in use. For instance, a SHM application may introduce strict requirements regarding network throughput since sensor nodes typically involved (e.g., accelerometers) may acquire data at high frequency. This is not the case with some proposed IoT-based smart agriculture applications that focus on maximizing the energy efficiency of battery-powered devices [59]. In this thesis, we focus on four main requirements of H-WMSNs:

- *Device Heterogeneity*. The current IoT landscape is characterized by extreme hardware/software fragmentation, which poses formidable interoperability issues. If we limit our attention to the radio communication, several M2M communications for WMSN are available on the market, each characterized by its protocol stacks and network performance. Existing UAV-aided systems rely on the assumption that both the network and aerial segments are based on the same M2M stack, limiting the potential and applicability of such systems. Vice versa, we can envision a more ambitious scenario in which different providers manage ground and aerial segments, and their integration has not been designed at design time. Similarly, we may envision generic UAV swarms with drones provided by different manufacturers and mapped to different M2M technologies.

- *Adaptive QoS management.* As mentioned before, IoT monitoring applications may have different QoS requirements. At the same time, in H-WMSN, static and mobile nodes may experience time-varying network performance on the air-to-air and air-to-ground links based on the scenario characteristics (e.g., propagation conditions) and on their current positions. The problem may become even more challenging in multi-radio environments, with additional parameters to be tuned. In our work, we address the problem of dynamic reconfiguration of H-WMSN nodes so that per-link QoS requirements are continuously monitored and met. This is addressed through the dynamic selection of M2M stack on each multi-radio link described in Chapter 6.

- *Reliability.* In IoT monitoring applications, Sensor or Sink nodes can fail for many reasons, including system errors or battery depletion. Also, in ground WMSN, nodes may also serve as relays and enable multi-hop communication towards the Sink. Hence, a single device failure may lead to network fragmentation and raise harmful issues such as communication disruptions, data loss, and reduced efficiency. Therefore, the design of a reliable and robust system capable of handling device failures is crucial. In this thesis, we investigate the usage of UAVs as recovery units able to temporarily replace the operations of ground devices till the intervention of a manual operators. Of course, such solution poses additional research challenges, such as where to place the recovery UAVs and how to detect the failure of ground devices in an effective manner.

- *Scalability.* IoT monitoring applications may include large sets of networked nodes sensing a wide scenarios. Thanks to their inner characteristics, such as multi-path support, WMSN has been designed to scale at least better than traditional, single-path/single-sink approaches. Still, we have to consider that new devices, on the ground or aerial network segments, may be dynamically added and that each device may offer different capabilities in terms of sensing (e.g., a novel sensor) or computational power. Hence, we believe that to unlock full network scalability, the existing routing strategies must be coupled with effective discovery mechanisms enabling each node to be aware of the services/functionalities offered by other peers. In Chapter 6, we demonstrate how such discovery may ease the federation of UAVs collaborating on a learning task.

The aforementioned requirements introduce several research challenges which must be considered at design time. In the thesis, we address four main challenges of H-WMSNs:

- *Interoperability.* Achieving radio interoperability is a complex task. If not done properly, the result may be a system that is difficult to manage and extend to support additional M2M stacks. As a result, the multi-radio environment may become a bottleneck rather than a system resource.

- *Mobility.* UAVs are mobile devices; hence the performance of the links they establish may not be stable. This creates challenges in maintaining consistent and reliable communication and can have a significant impact on the overall system performance.
- *Connectivity.* Due to the nodes' mobility and other factors, the mesh networks can experience link failures which may lead to network partitions. The latter may impact the system coordination and -in more severe cases- determine application disruption.
- *Power Constraints.* UAVs are typically powered by batteries, which have limited energy and lifetime, requiring careful power management to ensure the network's operational efficiency and reliability.

3.3 Research Objectives

The thesis aims to develop solutions for H-WMSN, considering the four requirements mentioned in the previous Section. Specifically, separate research contributions have been proposed for each of the three layers composing the H-WMSN:

1. **Ground WMSN**. The possibility of building ground WMSN has been test-fielded. After a preliminary literary review, during which different WMSN stacks have been analyzed and compared, we focused on the emerging BLE Mesh standard, which allows to deploy of multi-hop, short-range WSN on top of the popular BLE stack. Such technology envisions low-power performance and native multi-path routing support; hence it may fit well the requirements of IoT monitoring applications in harsh environments, such as SHM of remote structures. The goal of our research study (described in Chapter 4) was to characterize the profiles of IoT systems supported by the BLE Mesh stack, as well as test-field its usage for air-to-ground communication links.
2. **Aerial WMSN**. As second contribution of the thesis, we restricted our attention to the aerial WMSN collecting data from an isolated ground WMSN. Specifically, we addressed the problem of how to deploy the aerial nodes so that they are able to maximize the effectiveness of the data collection in terms of ground nodes connected and in QoS of the UAV-to-UAV and UAV-to-Sink links while maintaining the aerial mesh connectivity. A novel swarm mobility algorithm (called ECLIPSE chapter 5) was proposed and implemented: differently from existing solutions, ECLIPSE supports distributed swarm management relying on radio signal localization only (no GPS).

3. **Ground/Aerial WMSN integration**. Finally, we investigated hybrid scenarios with air-to-ground communication links among the two network meshes. Here, we investigated effective data collection strategies in the presence of multi-radio UAVs/Sink nodes. In addition, we addressed the reliability requirement by enabling the faulty node replacement.

For the second and third research objectives, a major contribution of thesis consisted in the design and implementation of the `Uhura` framework. The latter is a novel software framework that facilitates communication both within an aerial swarm and between the swarm and the ground devices, regardless of the multiple M2M communication technologies used. It includes a core module, known as the `Uhura Core`, which dynamically selects the most suitable adapter for each link based on the QoS needs of the application. The `Uhura Adapter` provides an abstraction layer for managing multiple M2M communication stacks. The `Uhura` framework has been extended with additional services, supporting scalability (see Section 6.6) and reliability requirements (see Section 6.5).

3.4 Methodologies

To achieve the objectives described above, several research methodologies have been used, including:

- Literature review: To characterize the current state of the art of mesh and UAV-aided networking and identify the main challenges and requirements for the proposed scenario.
- Software design: To develop the *Uhura* Framework that addresses the challenges and requirements identified in the literature review.
- Simulation and emulation: To evaluate the performance of components of the proposed framework using simulation tools such as `OMNet++` [1].
- Field experiments: To validate the proposed framework in a real-world scenario, such as a bridge structure, and to collect data on energy consumption, latency, throughput.
- Data analysis: To analyze the data collected from simulations, emulations, and field experiments and draw conclusions on the performance of the proposed framework.

[1] https://omnetpp.org/

Chapter 4

Ground Mesh Networking

In Chapter 3, we identified the first objective of this thesis as the development of ground WMSN solutions to support IoT monitoring applications in harsh environments. To achieve this goal, we conducted a literature review and focused on the BLE Mesh standard as a potential solution. In this chapter, we describe our research study on characterizing the profiles of IoT systems supported by the BLE Mesh stack and testing its usage for air-to-ground communication links. We also discuss our findings and the potential applications of this technology for ground WMSN.

Introduction

In this project, we explore the use of Bluetooth Mesh technology for monitoring indoor environments and mobile device localization. The focus of this study is on the BLE Mesh standard, which is promoted by the Bluetooth Special Interest Group (SIG) [60]. BLE Mesh is not a new wireless communication technology but rather a networking solution that enables multi-hop communication and routing on top of the legacy BLE stack. The advantages of BLE Mesh over other WPAN solutions, such as LoRa and Sigfox [61, 62], Zigbee and 6LoWPAN [63], include the ability to seamlessly integrate legacy BLE devices into the mesh network [64, 65] and the ability to exploit BLE advertisements for device localization and location-aware networking.

The research approach for this study is divided into three stages. The first stage involves designing an IoT-based monitoring system using BLE Mesh and a layered software stack running on an Edge Processing Unit (EPU). The second stage involves implementing the system in an indoor, single-floor test bed using the ESP-BLE-MESH framework by Espressif [66] and evaluating the system performance under different traffic loads and network sizes. The final stage involves discussing the suitability of BLE Mesh for indoor navigation

services [67] and identifying key challenges and limitations that need to be addressed to realize its potential fully.

Additionally, previous studies have raised concerns about the performance of multi-hop BLE Mesh networks and their ability to support IoT monitoring systems with specific Quality of Service (QoS) requirements [68, 69]. These studies highlight the number of parameters that need to be tuned, which can create complex trade-offs between energy efficiency and end-to-end delay, and the interference issues on the ISM bands. Furthermore, multi-hop deployments can suffer from broadcast problems caused by the controlled flooding mechanism, which simplifies the routing strategy but affects scalability and throughput [70]. In this project, we aim to provide insights into the capabilities and limitations of BLE Mesh technology for indoor monitoring and device localization by addressing research issues such as network scalability, traffic loads, and QoS requirements for different classes of IoT applications and services. By using a thorough and methodical approach, we hope to contribute to a better understanding of the potential of BLE Mesh for IoT monitoring systems and its capability to support the growing demands of IoT applications.

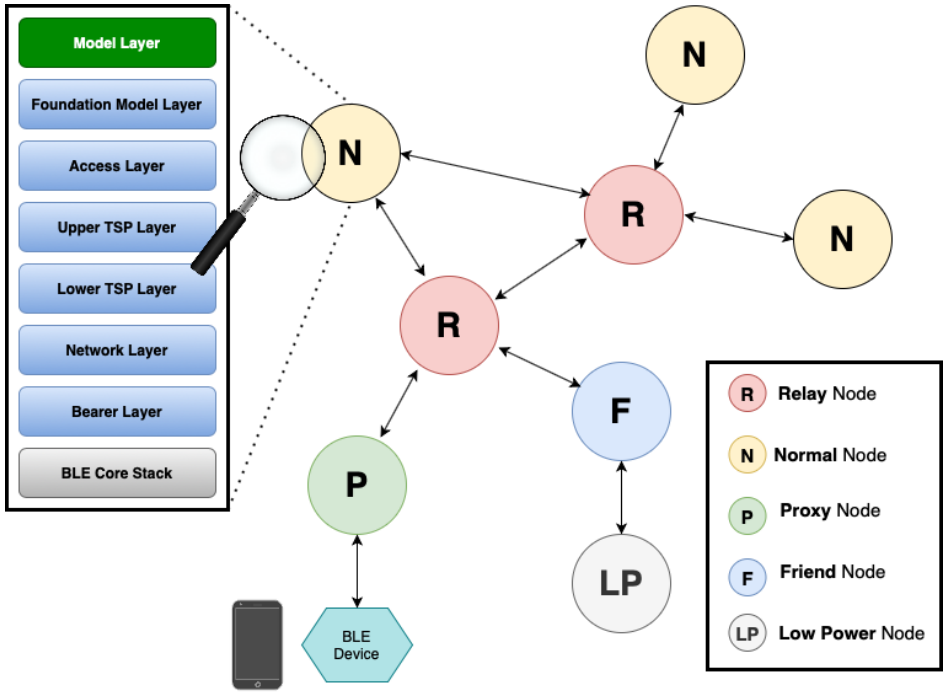


Fig. 4.1 The BLE Mesh-based monitoring system with the layered software suite installed on the EPU device (on the right) and the sequence messages involved in the localization procedure (on the left).

In addition to the above-mentioned research approach, we also consider the impact of different network sizes, densities, and traffic loads on the performance of BLE Mesh

networks. This will allow me to determine the scalability of the technology and its ability to support a large number of devices and data traffic. Furthermore, we also investigate the impact of different IoT application profiles and their corresponding QoS requirements on the BLE Mesh network. This will allow me to identify the classes of IoT applications and services that BLE Mesh can support and the level of QoS that can be achieved.

Additionally, we also consider the impact of different network topologies and deployment scenarios on the performance of BLE Mesh networks. This includes the impact of different node densities, node positions, and the number of hops on the overall performance of the network. Furthermore, we also investigate the impact of different routing strategies on the performance of BLE Mesh networks. This will allow me to determine the most suitable routing strategy for different deployment scenarios and IoT applications.

Overall, this project aims to provide a comprehensive evaluation of the performance and suitability of BLE Mesh technology for IoT monitoring systems and indoor navigation services. By investigating the key research issues mentioned above, we hope to provide valuable insights into the capabilities and limitations of BLE Mesh technology and contribute to a better understanding of the potential of BLE Mesh for IoT monitoring systems and its capability to support the growing demands of IoT applications.

4.1 System Design

Without loss of generality, we consider a multi-room, single-floor indoor scenario like the one depicted in Figure 4.1. The goal of this study is to investigate the design and deployment of BLE Mesh networking solutions with twofold functionalities: (i) *scenario monitoring*, i.e. the WSN is able to collect environmental data from BLE sensors spread out all over the scenario; (ii) *human monitoring*, i.e. the WSN enables room-level indoor localization of BLE devices moving within the same building. A possible use case is represented by an industrial environment, where there is a need of monitoring the state of health of critical equipment (e.g. the temperature of a tank) as well as of tracking the access of workers to the restricted area, for security or safety reasons. For this purpose, in Section 4.2, we assess the characteristics of IoT monitoring applications that can be supported by the BLE Mesh, by evaluating the system performance under different traffic loads and node density.
The proposed system includes many hardware/software components. On the hardware side, we installed $N+1$ BLE devices, $n_0, \ldots, n_N$, forming a connected multi-hop WSN; n_0 is used as data collector, also called Sink node in the following, and is directly connected to an Edge Processing Unit (EPU) through a Serial cable. All the BLE devices, except for the Sink, are equipped with multiple sensors and report their sensing data to the Sink, as further

detailed in Section 4.1.1. On the EPU, we deployed the software stack for data processing and analytics. More specifically, the *Parser Module* is in charge of filtering the messages received from the BLE Mesh network via the Sink node. The useful data are extracted and sent to the *Monitoring Module*, which includes two separate software components: (*i*) the *Network Monitor Module* is in charge of computing the network metrics of the BLE Mesh network as well as of extracting the application-specific features from the sensor data (e.g. the average of temperature values); (*ii*) the *Human Monitor Module* is in charge of detecting the presence of mobile devices, and of estimating their current positions according to the procedure detailed in Section 4.1.2. Finally, the *Data Module* allows to store network and application-related metrics and position data in a database, and to visualize them through a Web Dashboard powered by external tools. In the following, we detail the operations of BLE Mesh networking, and of the proposed BLE-based localization technique.

4.1.1 BLE Mesh Networking

Consider a system setup where all the BLE Mesh devices are configured as *Router* nodes. We do not analyze energy efficiency issues in the paper; hence all the nodes are assumed to be powered by current. Also, all the N nodes (Sink excluded) support the Sensor Model and act as Servers, i.e., they hold a state related to the current sensing values, while the Sink node acts as Client. The data collection process works as follows. We assume that the IoT monitoring system must collect measurements from each sensor at periodic intervals. Let T_p be the interval among consecutive sensing actions. Every T_p seconds, the EPU triggers the Sink node, which in turns publishes a new `GET` request on the `measure` group address (to which the other N Servers subscribed), and expects to receive N measurements from the other devices. The received messages are then transferred to the EPU via the Serial connection and here processed for the computation of the network metrics and for the sensor data analytics. The setting of T_p is clearly application-dependant and strongly affects the system performance, as further investigated in Section 4.2.

4.1.2 BLE Mesh Localization

The localization procedure through the BLE Mesh network is illustrated in the left part of Figure 4.1. We assume that each agent moving within the environment (being a person or a robot) is equipped with a mobile device provided with BLE connectivity and provisioned to operate over the BLE Mesh as a Normal node. Let j indicate its unique unicast address. We omit further details regarding the characteristics of the mobile device. Rather, we focus on its interaction with the nodes composing the BLE Mesh network. More specifically, we

set-up the system so that: (*i*) all the $N+1$ Mesh nodes subscribed to the `loc_req` group; (*ii*) the Sink node subscribed to the `user_detected` group. The mobile device periodically advertises its presence by publishing a message on the `loc_req` channel, with TTL set to 1, every $T_{presence}$ seconds. Let k_j be the unique sequence number of the advertising message sent by the mobile device j. The BLE Mesh nodes located in the transmitting range of the mobile device will detect the message; however, they will not forward it because of the TTL restriction. We denote them as Anchor nodes in Figure 4.1; since their positions are assumed static and known, the current position of the mobile device can be inferred via trilateration techniques further detailed in Section 4.2. Let $S(j,k_j)$ be the list of Anchor nodes for mobile device j sending the advertisement with sequence number k_j, where $|S(j,k_j)| \leq N+1$. Each node $s \in S(j,k_j)$ reports the event to the Sink node by publishing a message on the `user_detected` topic by adding the following information to the payload: $< ad_s, P(j,k_j), t_c >$ where ad_s is its unicast address, $R(j,k_j)$ is the Received Signal Strength (RSS) of the received message from the mobile node j, and t_c is the current time-stamp. The message is filtered by the Parsing Module of the EPA, and the payload is processed by the Human Monitor Module, which is in charge of estimating P_j, i.e., the current position of mobile node j. We consider two possible formats for P_j, with different spatial granularity levels, i.e.: (*i*) P_j is the room id, hence the localization procedure aims at detecting the current room where the mobile node is located, or (*i*) P_j is a continuous value representing the 2D relative coordinates with respect to a reference point (e.g., the top left angle of the building).

4.2 Implementation And Performance Evaluation

In this Section, we describe the implementation of the BLE Mesh network, and we evaluate the performance over multiple configurations.

We rely on the Espressif ESP32[1] devices (*wroom* and *wrover* versions) to setup the BLE Mesh nodes. Espressif provides an implementation of the BLE Mesh stack that can be deployed and customized via the provided Espressif IoT Development Framework[2]. For the experiments, we uploaded two different profiles on the BLE Mesh devices: (*i*) the Sensor module profile, enabling the device to send sensory data and to exchange control messages, as well as the usage of different topics depending on the experiment, and (*ii*) the Relay module profile enabling the routing and message forwarding capabilities. We set the transmission power $P_{\text{tx}} = -6$dBm and used the default configuration with no packet retransmissions and no message acknowledgements. As depicted in Figure 4.1, the Sink node n_0 is connected

[1] *https://www.espressif.com/en/products/socs/esp32*

[2] *https://github.com/espressif/esp-idf* - Used version v4.3

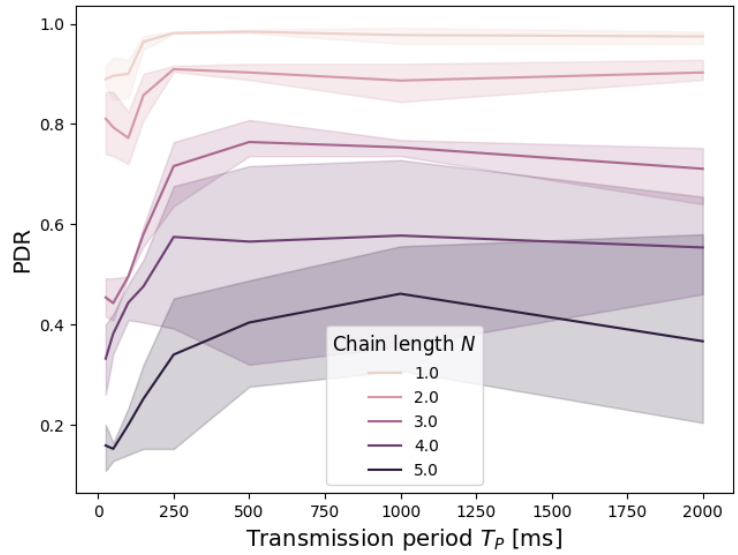


Fig. 4.2 The end-to-end delay varying T_P and N in the mesh topology scenario.

[

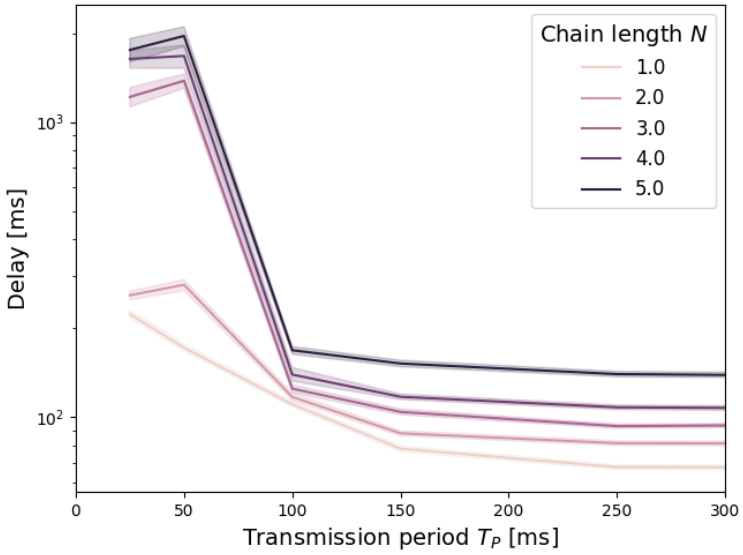


Fig. 4.3 The measurements for the path-loss calibration

directly to the EPU device. In our deployment, the Sink is constituted by a Raspberry Pi 3B+ executing Nodejs v16.4.0 scripts that implement the software modules (i.e., the Parser Module, the Data Module, the Network Monitor, and the Human Monitor) previously described.

First, we assess the network performance through the metrics computed by the Network Monitor Module, i.e., the Packet Delivery Ratio (PDR) and the end-to-end delay. The former states the network's reliability, while the latter indicates the crossing speed of data packets

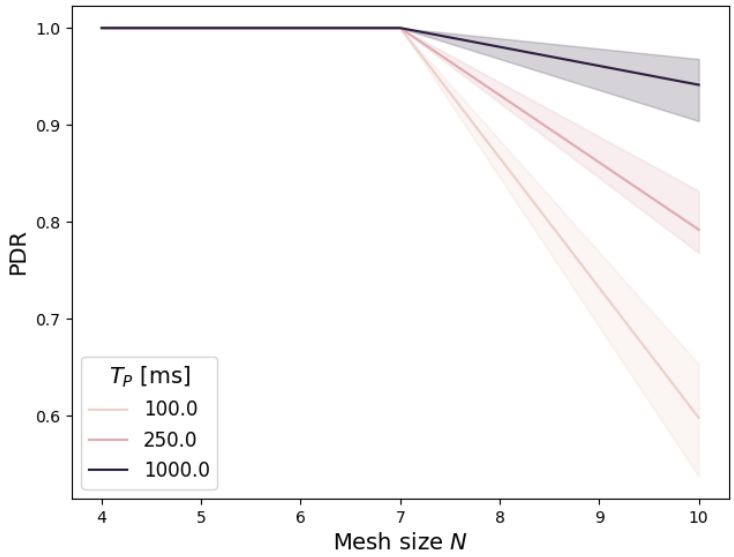


Fig. 4.4 The localization error for the different reference points $P_0, \ldots, P_8$.

through a multi-hop connection. Two different BLE Mesh network deployments have been considered: a chain topology and a multi-path mesh topology.
In the chain scenario, the $N+1$ nodes are placed at equal distances in such a way that device n_i is connected with its predecessor and its successor only, n_{i-1} and n_{i+1} respectively, with $1 \leq i < N$, while node n_N is the only sensor that is subscribed to the `measure` topic. In addition, we deployed the Relay profile module on all the devices, the Server Sensor profile only on n_N and the Client Sensor profile on n_0. Despite its simplicity, the topology permits us to analyze the multi-hop capabilities of the BLE Mesh network under self-interference and external interference conditions caused by WLAN devices operating on the same bands and on the same area. The evaluation results related to the chain topology are depicted in Figures 4.2 and 4.3. We varied the length of the chain N (denoted by different curves) and the transmission period T_P (denoted by different values on the x-axis). As expected, the chain length impacts negatively both the PDR (Figure 4.2) and the delay (Figure 4.3). For $N > 1$, the data packets experience a non-zero probability of packet loss, leading to a PDR value lower than 40% for $N = 5$. When increasing the number of hops, the variance of results becomes relevant due to the unpredictable fluctuations of the channel conditions. Similarly, the end-to-end delay increases accordingly with the chain length N and the related number of forwarding actions. The analysis in terms of the application-dependent variable T_P (on the x-axis) allows us to determine the maximum workloads of the BLE Mesh network. However, we can notice that the T_P parameter impacts the system performance only under high loads (low T_P), i.e., when many packet losses may be experienced due to self-interference issues or

buffer overflow events at the intermediate nodes. Figure 4.3 shows that the delay is higher than 1 second for $N = 5$ and $T_P \leq 50$ ms.

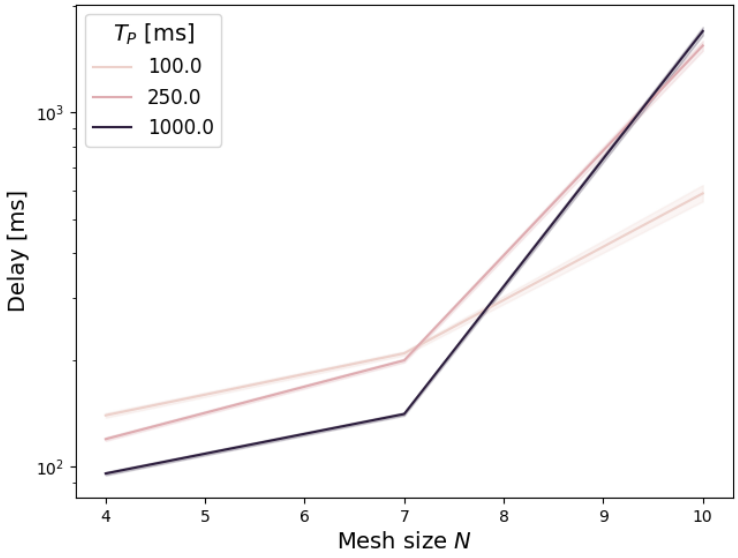


Fig. 4.5 The end-to-end delay varying T_P and N in the mesh topology scenario.

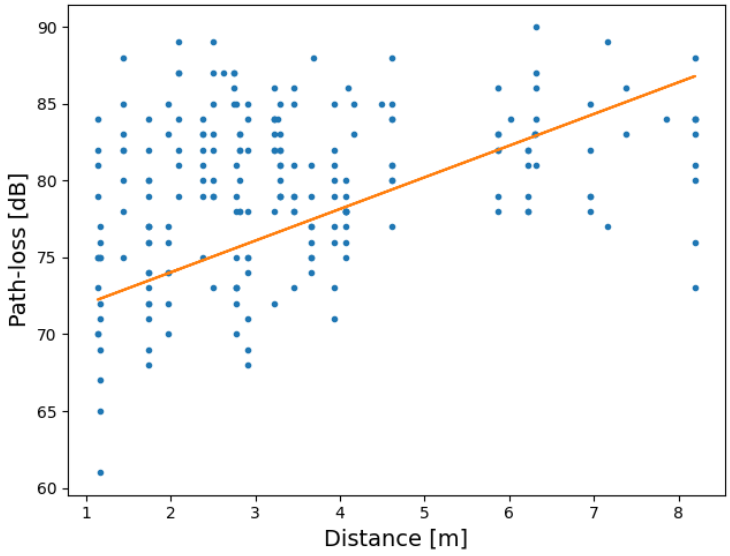


Fig. 4.6 The measurements for the path-loss calibration

The second deployment consists of a generic multi-path mesh topology, where the BLE devices are placed randomly within the scenario. Also in this case we analyzed the network metrics by varying the mesh size (N) and the transmission period (T_P). However, differently from the previous experiment, all the devices feature both the Relay and the Server Sensor profiles, while the Sink node features the Client Sensor profile only. All the nodes but the sink

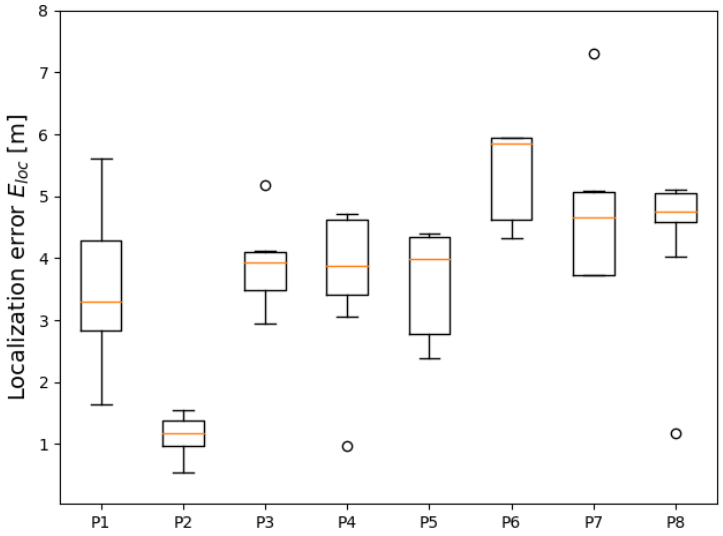


Fig. 4.7 The localization error for the different reference points $P_0, \ldots, P_8$.

subscribe to the `measure` topic. Figures 4.4 and 4.5 show respectively the PDR and delay metrics. Regarding the PDR index, we can notice a significant performance degradation with a mesh of 11 nodes ($N = 10$). It is easy to notice that the T_P parameter becomes effective only when $N \geq 8$, suggesting that the network size, and mainly the flooding mechanism, impacts the system performance more than the network load. Similar considerations can be drawn from the end-to-end delay results shown in Figure 4.5.

Finally, we analyze the performance of the Human Monitor Module. Figure 4.8 shows the planimetry of the indoor environment used for our localization tests, i.e. a floor hosting the research laboratory at the Department of Computer Science and Engineering of the University of Bologna. We consider $N = 12$ nodes (denoted as red squares and green diamonds in the Figure) and $P = 8$ localization points (the blue dots) where the experiments are executed. The mobile user j is provided with a provisioned BLE Mesh node and walks through predefined fixed points ($P_1, \ldots P_8$ in Figure 4.8) where the current position is estimated and the localization error is computed. As described in Section 4.1.2, the mobile device sends a message on the `loc_req` topic with TTL $= 1$, and every BLE Mesh node that receives the message acts as an Anchor node. Then, it computes the RSS value and forwards it to the Sink node. The Human Monitor Module implements the *Least-Squares Trilateration Algorithm* to estimate P_j from the received RSS measures. It is worth remarking that the aim of this work is not to evaluate the localization algorithm itself, rather to evaluate the ability of the BLE Mesh to support it. Interested readers can refer to [71] for other indoor localization algorithms that could be deployed on top of our testbed.

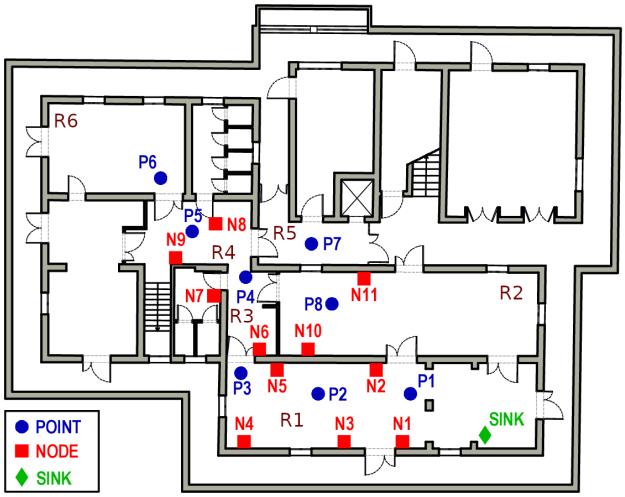


Fig. 4.8 Scenario map used for the localization deployment where $N = 11$. The red squares and the green diamond represent the devices belonging to the BLE Mesh; the blue points indicate the locations where the localization tests have been carried out.

The Trilateration Algorithm estimates the device location assuming the knowledge of distances from known anchor points. In our case, the distance d between the mobile device and the Anchor node n_i is computed from the RSS value by using the log-normal path-loss model: $d = d_0 \cdot 10^{\frac{P_{tx} - RSSI - PL_0}{10 \cdot \alpha}}$, where $d_0 = 1$ is the close-in reference distance at which the path-loss PL_0 has been measured and α is the path-loss exponent. The values of PL_0 and α have been empirically calibrated as shown in Figure 4.6. The accuracy of the localization process is depicted in Figure 4.7; the points on the x-axis $(P_0, \ldots, P_8)$ indicate the reference measuring locations depicted in the map of Figure 4.8. The y-axis shows the localization error E_{loc}, in meters, as the difference between the estimated position computed by the Human Monitor Module and the ground truth.

We can notice significant differences in the E_{loc} values between the locations close to the Sink node (e.g. P_2) and the others (e.g. P_6, P_7, P_8). At location P_2, $E_{\text{loc}} \approx 1$m due to the presence of many Anchor nodes and the proximity to the Sink. Vice versa, locations P_6, P_7, P_8 experience higher E_{loc} values because they are placed on the borders of the mesh network. From further investigations, we realized that the poor localization accuracy is again a consequence of the poor network performance; indeed, in Figure 4.2 we show that the PDR sharply decreases with the number of hops. As a result, even if the presence of a mobile node is detected by some Anchor nodes, only few messages belonging to group `user_detected` are reaching the Sink node, hence negatively affecting the calculations of the trilateration algorithm. The average localization error is around 4m, hence this system can be considered suitable only for low-granularity spatial requirements. Indeed, we

implemented a room localization module that exploits the knowledge of the floor planimetry for the estimation. Figure 4.9 depicts the per-room accuracy of the localization process, while the red line indicates the average accuracy within the target building. Similarly to the previous experiment, the accuracy is highly dependant on the user position with respect to the Sink; the accuracy exceeds 80% for rooms close to it, with the exception of room R_3 which is actually a very small corridor. On the other side, the accuracy drops drastically for rooms covered by few Anchor nodes (since the trilateration algorithm lacks of inputs), and for rooms far away from the Sink due to the poor network performance.

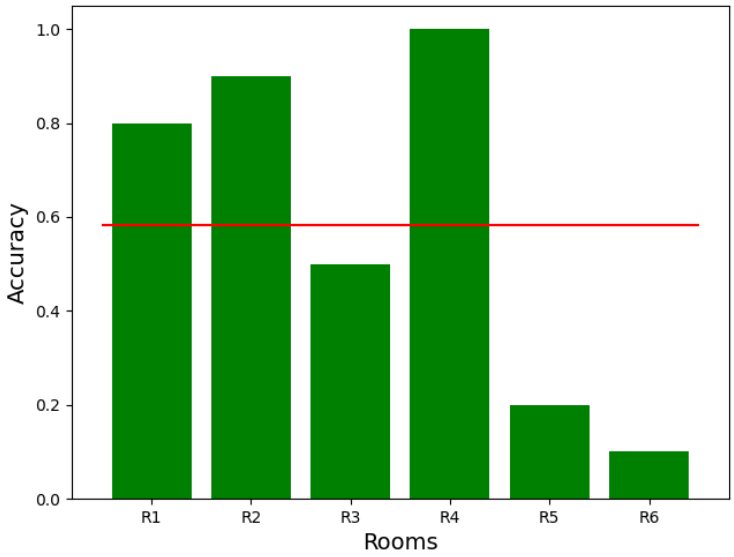


Fig. 4.9 The room detection accuracy at different rooms of the scenario. The red line indicates the average accuracy within the building.

Based on these results, we can conclude that the proposed BLE Mesh network is not capable of supporting IoT applications with strict delivery requirements, or where the data aggregation/fusion algorithm is computed on the Sink node. Also, high transmission frequency impacts harmfully the system performance, but still less than the network size.

Chapter 5

Aerial Network Deployment

Chapter 3 identified the second objective of this thesis as the development of an aerial WMSN to collect data from an isolated ground WMSN. To address this problem, we proposed and implemented a novel swarm mobility algorithm called ECLIPSE. Unlike existing solutions, ECLIPSE supports distributed swarm management relying on radio signal localization only, without the need for GPS. In this chapter, we describe the design and implementation of ECLIPSE, as well as our experiments and evaluation of its performance in maximizing data collection effectiveness while maintaining aerial mesh connectivity.

Introduction

In this project, we consider a generic scenario composed of isolated, mobile ground nodes (MGNs), and we investigate the deployment of distributed aerial mesh networks aimed to provide wireless coverage of the target area and multi-hop connectivity between ground nodes. To this aim, we propose ELAPSE, a distributed UAV swarm architecture that addresses mobility-related (e.g., aerial connectivity), task-related (e.g., ground coverage), and networking-related functionalities. Differently from the literature, the ELAPSE framework takes into account the QoS on the AtA and AtG links both during the network formation and maintenance in order to dynamically meet the application requirements of the ground nodes. In addition, the ELAPSE framework does not rely on geo-localization capabilities of the aerial/ground devices; hence, it can be deployed on mini-drones (not provided with the GPS sensor) or it can support aerial mesh formation and mobility on environments not covered by the GPS (e.g., indoor scenarios). More in detail, three main contributions are described in this paper:

- We extend the QoS-aware UAV swarm mobility model in [72] for disaster environments: the distributed algorithm aims to maximize the number of connected MGNs while guaranteeing the quality of the AtA and AtG links. In addition, it ensures mesh connectivity and collision avoidance among the UAVs.
- We investigate the implementation of the proposed swarm mobility algorithm in scenarios where geo-location capabilities are not available: to this aim, two distributed neighbor localization schemes are presented, one based on local sensor data processing (e.g., IMU and Wi-Fi RSSI values) and the other on cooperative mechanisms. The operations of the proposed algorithms are further enhanced by means of error filters and computational swarm intelligence techniques.
- We validate the proposed solutions through a twofold evaluation. We consider large-scale OMNeT++ simulations of the ELAPSE framework and measure its performance in terms of localization accuracy and coverage under varying node density conditions. In addition, we validate the effectiveness of the QoS-aware mechanism on a small ground robotic test bed.

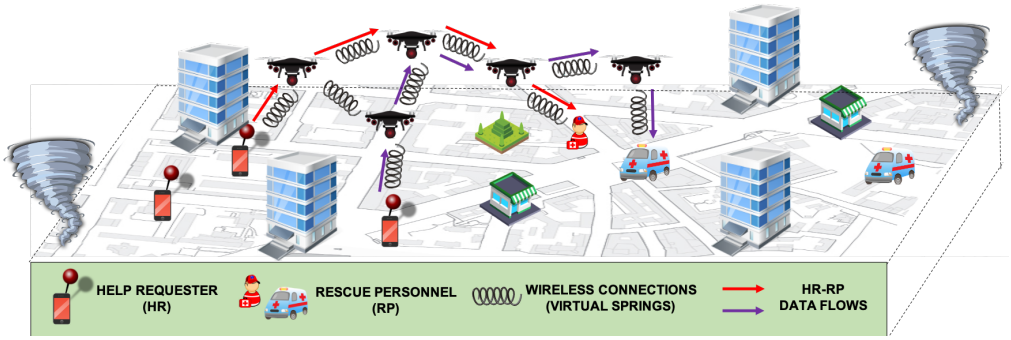


Fig. 5.1 The scenario considered in this study and the deployed framework.

5.1 System Model

5.1.1 Scenario definition

We consider the emergency scenario of size $M \times M$ depicted in Figure 5.1. Let $U = \{u_1, u_2, \ldots u_{N_U}\}$ be the set of the available UAVs flying at a fixed altitude f_a from the ground. Similarly, let $G = R \cup H$ be the set of Mobile Ground Nodes (MGNs), further divided into two sub-sets: rescue personnel $R = \{r_1, r_2, \ldots r_{N_R}\}$ and help requesters $H = \{h_1, h_2, \ldots h_{N_H}\}$. The system evolves over ordered time $T = \{t_0, t_1, \ldots\}$ having time slots of length equal to

t_{slot} seconds. We denote with $\mathbf{p}_{n_i,k}$ the current absolute 3D (error-free) position of node n_i in the scenario at time t_k, with $n_i \in N = U \cup G$. Localization errors are introduced in Section 5.3. Each help requests $h_i \in H$ is provided with a mobile app, through which it generates a data rate equal to $D(h_i)$ bps; the destination of the flow originated by $h_i \in H$ is any $r_j \in R$ available in the scenario. We abstract from the specific content of the communication between h_i and r_j, which might include any emergency-related information e.g. current position, video, audio, etc, that might support the rescue operations. The goal of the ELAPSE framework is to enable the communication between help requesters and rescue personnel via multi-hop aerial links, by maximizing the number of data-flows which are currently served. To this aim, the optimal placement of the UAVs must be determined.

We modeled the system as a complete graph $\mathscr{G} = \{N, E\}$, where $N = U \cup G$ is the set of nodes and $E = \{\langle n_i, n_j \rangle, \dots\}$ is the set of edges. Let $C^k(n_i, n_j)$ the capacity at time slot t_k of the wireless AtG/AtA link between nodes $n_i, n_j \in N$: the value of $C^k(n_i, n_j)$ depends on the wireless propagation model that is introduced later in Section 5.1.3. By construction, $C^k(n_i, n_j) = 0$, $\forall n_i, n_j \in G$, i.e. no direct communication between ground nodes is possible. We model the communication network topology through the following state variables:

- $e^k(n_i, n_j)$ indicates whether nodes $n_i, n_j \in N$ are connected by a wireless link at time slot t_k, i.e. $C^k(n_i, n_j) > 0$: in such case, $e^k(n_i, n_j)$=1, $e^k(n_i, n_j)$=0 otherwise.
- $\hat{e}^k(n_i, n_j)$ indicates whether there exists a multi-hop connection between nodes $n_i, n_j \in N$, i.e. $\exists\ \text{path}^k_{i,j} = \{n_1, n_2, \dots, n_{|\text{path}^k_{i,j}|}\} \subseteq N$ such that $n_1 = n_i$, $n_{|\text{path}^k_{i,j}|} = n_j$, and $e^k(n_q, n_{q+1}) = 1$, $\forall q < |\text{path}^k_{i,j}|$.

Similarly, we introduce the following state variable in order to model the multi-hop connectivity through the aerial mesh:

- $l^k(n_i, n_j, h_z)$ indicates whether the link between nodes $n_i, n_j \in N$ is used to convey the traffic flow from n_i to n_j originated by the help requester $h_z \in H$ during time slot t_k: in such case, $l^k(n_i, n_j, h_z)$=1, $l^k(n_i, n_j, h_z)$=0 otherwise.

5.1.2 Problem formulation

Based on the definitions above, the research problem can be formulated as follows:

$$\textbf{find} \quad \vec{p}_{u_i,k} \quad \forall u_i \in U \tag{5.1}$$

$$l^k(n_i,n_j,h_z) \quad \forall n_i,n_j \in N, h_z \in H \tag{5.2}$$

$$\textbf{maximize} \quad \sum_{u_i \in U, r_j \in R, h_z \in H} l^k(u_i,r_j,h_z) \tag{5.3}$$

subject to the following constraints:

$$\sum_{u_i \in U, r_j \in R} e^k(u_i,r_j) \geq 1 \tag{5.4}$$

$$\overset{*}{e}{}^k(u_i,u_j) = 1 \quad \forall u_i,u_j \in U \tag{5.5}$$

$$\text{dist}^k(u_i,u_j) \geq \text{dist}_{\min} \quad \forall u_i,u_j \in U \tag{5.6}$$

$$\sum_{h_z \in H} \left(l^k(n_i,n_j,h_z) \cdot D(h_z) \right) \leq C^k(n_i,n_j) \quad \forall n_i,n_j \in N \tag{5.7}$$

$$l^k(u_i,u_j,h_z) \cdot \left(1 - \sum_{\substack{n_{i'} \in N \\ n_{i'} \neq u_i}} \left(l^k(n_{i'},u_i,h_z) \right) \right) = 0$$
$$\forall u_i,u_j \in U, h_z \in H \tag{5.8}$$

$$\sum_{n_i \in N} l^k(n_i,n_j,h_z) \leq 1 \quad \forall n_j \in N, \forall h_z \in H \tag{5.9}$$

$$\sum_{n_j \in N} l^k(n_i,n_j,h_z) \leq 1 \quad \forall n_i \in N, \forall h_z \in H \tag{5.10}$$

$$\sum_{n_j \in N, h_z \in H} l^k(r_i,n_j,h_z) = 0 \quad \forall r_i \in R \tag{5.11}$$

$$\sum_{n_j \in N} l^k(h_i,n_j,h_i) \leq 1 \quad \forall h_i \in H \tag{5.12}$$

$$\sum_{\substack{n_j \in N \\ h_z \in H, h_z \neq h_i}} l^k(h_i,n_j,h_z) = 0 \quad \forall h_i \in H \tag{5.13}$$

where the constraints are assumed to be $\forall t_k \in T$.

Here, constraint (5.4) states that the aerial mesh should connect at least one rescue personnel; constraint (5.5) ensures the aerial mesh connectivity, i.e., there must exist a multi-hop path

between each couple of UAVs; constraint (5.6) avoids the collisions among the UAVs; constraint (5.7) ensures that the throughput of each wireless link (AtA or AtG) cannot exceed the current link capacity; constraint (5.8) ensures the consistency of the data flow through the aerial mesh, i.e., if an aerial link u_i, u_j carries data originated by $h_z \in H$, then there is an incoming data link on UAV u_i (u_i is routing data for h_z); constraints (5.9) and (5.10) guarantee that there is no multipath in the mesh, namely, each data flow can be sent and received by only one node, respectively; constraint (5.11) states that the rescue personnel is not transmitting data; constraints (5.12) and (5.13) ensure that any data flow labeled with $h_i \in H$ is generated only by h_i.

5.1.3 Channel Model

We assume a generic path loss model on the AtA/AtG links, defined as follows:

$$PL(d) = 10 \cdot \alpha_{LT} \cdot \log_{10}(d) + \kappa_{LT} \tag{5.14}$$

where d is the node distance, α_{LT} is a decay model depending on the Link Type (AtA or AtG), and κ is a constant value related again to the frequency and to the Link Type in use. In this work we do not focus in the specific characterization of the AtA and AtG links. Instead, we used the generic log-distance path loss model by using different α_{LT} parameters. Readers can refer to [73] for an in-depth modeling of the AtG link. The values of the parameters used in the simulation study are reported in Section 5.4.2. The maximum capacity $C^k(n_i, n_j)$ between node n_i and n_j placed at distance d can be derived according to the well-known Shannon law:

$$C^k(n_i, n_j) = BW \cdot \log_2 \left(\frac{PT - PL(\text{dist}^k(n_i, n_j))}{\kappa^k_{\text{noise}}} \right) \tag{5.15}$$

where BW is the channel bandwidth, PT is the node transmitting power (both values are assumed constant for all nodes and wireless links), and κ^k_{noise} the current noise value. In this study, we aim to monitor the per-link Quality of Service (QoS) by means of the Link Budget (LB) metric. The latter is defined as the residual capacity of link between n_i (receiver) and n_j (transmitter) at time t_k, and it is computed as follows:

$$LB^k(n_i, n_j) = PR^k(n_i, n_j) - RS(n_i) \tag{5.16}$$

where $PR^k(n_i, n_j)$ is the received power at node n_i and $RS(n_i)$ is its receiving sensitivity that is specific of the wireless network interface. The LB metric measures the communication reliability, and it indicates when the link is going to break.

5.2 ELAPSE: Swarm Mobility Algorithm

The optimization problem presented in the previous Section requires a global knowledge of the scenario and a fine-grained description of the system evolution at each time slot. Given the practical limitations posed by a centralized approach, we propose here a distributed, iterative technique which continuously updates the UAV positions' with the aim of addressing the following requirements at the final deployment: (*i*) there is always at least one rescue personnel connected to the aerial mesh; (*ii*) the number of help requesters connected to the mesh is maximized; (*iii*) the aerial mesh is connected (i.e., no UAV clusters are created) but at the same time (*iv*) all the AtA and AtG links meet the QoS requirements expressed through the requested *LB* values.

For these purposes, we refine the the virtual spring model described in [74] and further extended in [75, 72] for channel-aware QoS support. For a comprehensive review of different deployment algorithms for UAV networks, reader can refer to [76]. More specifically, we assume that at each time slot $t_k \in T$, multiple virtual forces can act on each $u_i \in U$, i.e. $\vec{F^k_{i,1}}$,$\vec{F^k_{i,2}}$, ...,$\vec{F^k_{i,N_f}}$. Let $\vec{F(i)}$ be the sum of virtual forces acting on u_i, i.e.:

$$\vec{F(i)} = \sum_{j=0}^{N_f} \vec{F_{i,j}} \tag{5.17}$$

As depicted in Figure 5.1, we consider three virtual Forces Types (FT), i.e.: (*i*) Mesh-to-Mesh (MtM) forces, acting between two UAVs, (*ii*) Mesh-to-Helpers (MtH) forces, acting between an UAV and a help requester on the ground and (*iii*) Mesh-to-Rescuers (MtR), acting between and UAV and a member of the rescue teams. The MtM forces guarantee the internal connectivity of the aerial mesh, while the MtH/MtR forces enable space exploration and connectivity toward the ground nodes. Regardless of their type, all the virtual forces are modeled according to the well-known Hooke's law assuming that the force is proportional to the spring deformation [77]:

$$\vec{F}(\vec{x}_u, x) = \vec{x}_u \cdot \big(-k(FT) \cdot (x - l_0) \big) \tag{5.18}$$

where $k(FT)$ is the stiffness constant (assuming different values according to the force type, i.e. MtM, MtH or MtR), $\vec{x}_u$ denotes the spring unit-vector direction, x denotes the spring actual length, l_0 its natural length, and $\delta = (x - l_0)$ defines the spring displacement. Here we assume that $k(MtM)$ and $k(MtH)$ are constant values, which must be statically configured before the system deployment; Section 5.4 reports the values used in the experiments. Vice

versa, the $k(MtR)$ value is dynamically set by each UAV u_i according to the Equation below:

$$k_i^{MtR} = \begin{cases} KT & \text{if } \sum_{j \in NB_i} Cov^R(u_j, t_k) = 1 \\ k(MtH) & otherwise \end{cases} \tag{5.19}$$

Here, $Cov^R(u_j, t_k)$ returns the number of rescue personnel connected to UAV u_j at time-slot t_k, while NB_i denotes the list on 1-hop neighbors of UAV u_i. The stiffness value is set to $KT >> k(MtH)$ in case the UAV is the only node in its neighborhood providing connectivity to a rescue personnel, and hence it should avoid breaking the link. Vice versa, the MtR virtual spring behaves like the MtH one. Like in [72], we formulate the link displacement as a function of the requested and current LB on the $n_i - n_j$ link, i.e.:

$$\delta^k(n_i, n_j) = \sqrt[\alpha_{LT}]{\frac{\max(LB^k(n_i, n_j), LB^k_{\text{req}}(n_i, n_j))}{\min(LB^k(n_i, n_j), LB^k_{\text{req}}(n_i, n_j))}} - 1 \tag{5.20}$$

Here, $\delta^k(n_i, n_j)$ is the spring displacement, α_{LT} is the propagation decay exponent of Equation (5.14), again assuming different values based on the AtA (MtM) or AtG (MtH and MtR) links. Let $LB^k_{\text{req}}(n_i, n_j)$ be the requested link budget on the link connecting node n_i with node n_j at time slot t_k; $LB^k_{\text{req}}(n_i, n_j)$ represents the per-link QoS requirement, and it is computed based on maximum number of data-flows that can be supported. This value can be derived from Equations (5.15) and (5.16) as follows:

$$LB^k_{\text{req}}(n_i, n_j) = \left(2^{LL^k(n_i, n_j)/BW} - 1\right) \cdot \kappa^k_{\text{noise}} - RS(n_i) \tag{5.21}$$

Here, $LL^k(n_i, n_j)$ is the requested load on this specific link. In order to fulfill the constraint defined in Equation (5.7), it is defined as follows:

$$LL^k(n_i, n_j) = \sum_{h_z \in H} \left(l^k(n_i, n_j, h_z) \cdot D(h_z)\right) \tag{5.22}$$

The value $LB^k_{\text{req}}(n_i, n_j)$ of Equation (5.21) as a function of the required link load (Equation (5.22)). Figure 5.2 shows the trend of Equation (5.21) when varying the requested link load $LL^k(n_i, n_j)$. As described before, the Hooke's law defines the attractive force as well as the repulsive force. However, for the AtG links the UAVs do not need to be repulsed from ground nodes and hence we activate the MtH and MtR forces only if they are attractive.

Every t_{dec} intervals, each u_i computes the resultant force $\vec{F(i)}$ on Equation 5.17, and moves in the direction indicated by $\vec{F(i)}$, with constant speed. In [72], we assumed that the direction could be directly derived by assuming that the UAVs are equipped with any

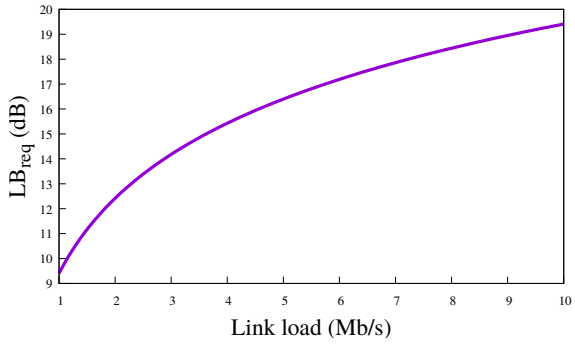


Fig. 5.2

geo-location system (like GPS, GLONASS, Galileo, etc) and that the position information is periodically exchanged among the UAVs. Here, we extend the study by removing such assumption; vice-versa, we address the case where each UAV and MGNs $n_i \in N$ is equipped only with Inertial Measurement Unit (IMU) sensors to estimate its actual velocity. Details on the how the UAV direction is estimated are provided in the Section below.

5.3 ELAPSE: Positioning Technique

In this study, we do not aim at estimating the absolute position of the UAVs and the MGNs (e.g. geo-location), rather their relative positions since only this information is relevant to compute the direction of each virtual spring according to Equation 5.17. To this aim, let $\vec{v}_{n_i,k}$ and $\vec{p}_{n_i,k}$ be the real velocity and position of each node $n_i \in N$ at time slot t_k, respectively. Also, let $\vec{p}_{(n_i,n_j),k} = \vec{p}_{n_j,k} - \vec{p}_{n_i,k}$ be the real relative position of node n_j with respect of node n_i and $\dot{\vec{p}}_{(n_i,n_j),k}$ be its estimated value computed by node n_i. For ease of exposition, we use the dot ($\dot{}$) notation to refer to an estimated value, which might differ from the real one.

While moving, each node $n_i \in N$ broadcasts one $\texttt{HELLO}_{n_i,k}$ message every $t_{\text{broadcast}}$ time slots, by including the following information:

$$\left\langle n_i, \dot{\vec{v}}_{n_i,k}, \dot{\vec{v}}_{n_i,k^-} \right\rangle \tag{5.23}$$

Here $\dot{\vec{v}}_{n_i,k}$ is the instantaneous velocity of node n_i at time t_k, while $\dot{\vec{v}}_{n_i,k^-}$ is the average velocity of node n_i between time slot $t_{k-t_{\text{broadcast}}}$ and t_k, defined by $\dot{\vec{v}}_{n_i}(k - t_{\text{broadcast}}, k)$. The function $\dot{\vec{v}}_{n_i}(k,k')$ returns the estimated average velocity of node n_i within the interval $[t_k..t_{k'}]$. After receiving an $\texttt{HELLO}_{n_j,k}$ message from node u_j, node u_i determines the Received Signal Strength (indicated as $\texttt{RSS}_{n_j,k}$ in the following) in order to estimate the distance from the sender node. We skip details on this issue, well addressed in the literature; interested

readers can refer to [78] for a comprehensive survey on the topic. We just highlight that, for evaluation purposes, we assume the presence of an additive white Gaussian noise, i.e. $\dot{\text{dist}}^k(u_i,u_j) = \text{dist}^k(u_i,u_j) + \mathcal{N}(0,\sigma_d)$, where σ_d is a system variable denoting the distance estimation error, whose impact on system operations has been evaluated in Section 5.4.

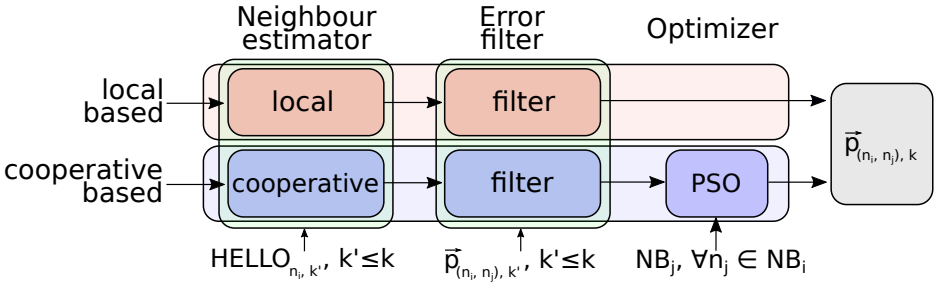


Fig. 5.3 The building blocks of the proposed relative localization algorithm.

The proposed relative localization algorithm involves three blocks executed sequentially, as depicted in Figure 5.3:

- **Neighbour estimator**: this module is executed by each node and it aims at estimating the relative position of its neighbours. To this purpose, two different methods are presented: a local algorithm (Section 5.3.1) and a cooperative one (Section 5.3.2). In both cases, the estimation exploits the RSS values and payloads of the $\texttt{HELLO}_{n_j,k}$ messages exchanged among the ELAPSE nodes.
- **Error filter**: this module applies proper filters on the output of the previous step, in order to avoid large variations among consecutive estimations (Section 5.3.3). To this purpose, a per-node estimation history is adopted.
- **Optimizer**: this module is executed only in cooperative mode and it further improves the position estimation by taking into account additional information from each neighbour node (Section 5.3.4).

5.3.1 Local Neighbour Estimator

We consider a local technique wherein each node n_i estimates the position of its neighbor n_j by using the information contained in the $\texttt{HELLO}_{n_j,k}$ message. At each message reception, node n_i performs two, independent estimation of its distance from n_j, i.e.: (*i*) it derives the distance from the $\texttt{RSS}_{n_j,k}$ value and (*ii*) by considering the average nodes' speed and the temporal interval among consecutive message receptions. By computing the intersection

of the circumferences associated to the estimated distances, and under the assumption of 2D mobility (since all the UAVs are flying at the same altitude), two different solutions for the n_j position at time t_k are derived and inserted into a solution set $P_{(n_i,n_j),k}$. After having collected a threshold number of solutions, the *Z-score* function is applied on $P_{(n_i,n_j),k}$ in order to remove the outliers; finally, the centroid is returned as the estimated position of n_j. The process above is iterated by n_i for all its active neighbours. This latter is defined as the set of nodes from which n_i has received at least one `HELLO` message within a timeout interval.

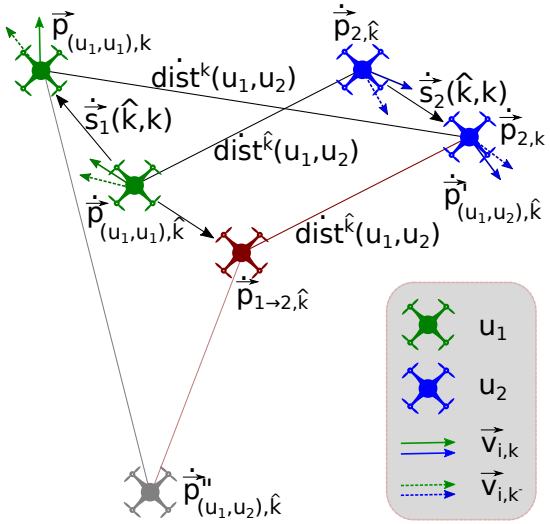


Fig. 5.4 One-step position estimation.

More formally, at each reception of the $\texttt{HELLO}_{n_j,k}$ sent by node n_j, the node n_i updates the RSS-based position estimation $\dot{\vec{p}}_{(n_i,n_j),k}$ for node n_j. Then it stores such value in a local list $\texttt{PKT}^{i,j}_{\text{rcv}}$ along with the values contained in the $\texttt{HELLO}_{n_j,k}$ message, i.e.: $\texttt{pkt}^{i,j}_{k} = \left\langle k, \texttt{RSS}_{n_j,k}, \dot{\vec{v}}_{n_j,k}, \dot{\vec{v}}_{n_j,k^-}, \dot{\vec{p}}_{(n_i,n_j),k} \right\rangle$. Given the dynamic nature of the aerial mesh topology, we consider dynamic update mechanisms of the data structures, i.e. node n_i will remove the $\texttt{PKT}^{i,j}_{\text{rcv}}$ list in case no packet from node n_j has been received within a time threshold t_{timeout}. We can hence define the active neighbours of node n_i as: $\text{NB}_i = \{n_j \in N \mid \texttt{PKT}^{i,j}_{\text{rcv}} \neq \emptyset\}$.

Algorithm 1 shows the pseudo-code the proposed technique. In the main loop (lines 6 - 16), the solution set $P_{(n_i,n_j),k}$ is updated, by considering the $\texttt{HELLO}_{n_j,\hat{k}}$ message received at time slot $t_{\hat{k}}$, with $\hat{k} \leq k$. We consider a coordinate system rooted at u_i at time slot t_k, i.e. $\vec{p}_{(n_i,n_i),k} = (0,0)$. The process of position estimation that is executed at each loop cycle is described in Figure 5.4.

Algorithm 1: The LOCAL estimation algorithm

Input: $\mathtt{PKT}^{i,j}_{\mathrm{rcv}}$

Function $\mathtt{Identity}(n_j, \vec{p}1, \vec{p}2)$**:**

  return $\{\vec{p}1, \vec{p}2\}$

Function $\mathtt{OnPacketRCV}(\mathit{HELLO}_{n_j,k})$**:**

  $P_{(n_i,n_j),k} \leftarrow \emptyset$; $k' \leftarrow k$;

  $\dot{\vec{v}}_j \leftarrow (0,0)$; $\vec{p}_{(n_i,n_i),k} \leftarrow (0,0)$;

  forall $\mathit{pkt}^{i,j}_{\hat{k}} \in \mathit{PKT}^{i,j}_{\mathrm{rcv}}$ ***descending ordered by*** $\hat{k}$ **do**

    $\dot{\vec{s}}_i(\hat{k},k) \leftarrow \dot{\vec{v}}_{n_i}(\hat{k},k) \cdot (k-\hat{k}) \cdot t_{\mathrm{slot}}$

    $\dot{\vec{p}}_{(n_i,n_i),\hat{k}} \leftarrow \vec{p}_{(n_i,n_i),k} - \dot{\vec{s}}_i(\hat{k},k)$

    $\dot{\vec{v}}_j \leftarrow \frac{\left(\dot{\vec{v}}_{j,k'-} \cdot t_{\mathrm{slot}} \cdot (k'-\hat{k})\right) + \left(\dot{\vec{v}}_j \cdot (k-k')\right)}{k-\hat{k}}$

    $\dot{\vec{s}}_j(\hat{k},k) \leftarrow (k-\hat{k}) \cdot t_{\mathrm{slot}} \cdot \dot{\vec{v}}_j$

    $\dot{\vec{p}}_{i \to j,\hat{k}} \leftarrow \dot{\vec{p}}_{(n_i,n_i),\hat{k}} + \dot{\vec{s}}_j(\hat{k},k)$

    $\left\langle \vec{p}'_{(n_i,n_j),\hat{k}}, \vec{p}''_{(n_i,n_j),\hat{k}} \right\rangle \leftarrow \mathtt{CI}\left(\vec{p}_{(n_i,n_i),k}, \mathrm{dist}^{k}(n_i,n_j), \dot{\vec{p}}_{i \to j,\hat{k}}, \mathrm{dist}^{\hat{k}}(n_i,n_j) \right)$

    $f_{n_j,k} \leftarrow \mathtt{Identity}(n_j, \vec{p}'_{(n_i,n_j),\hat{k}}, \vec{p}''_{(n_i,n_j),\hat{k}})$

    $P_{(n_i,n_j),k} \leftarrow P_{(n_i,n_j),k} \cup f_{n_j,k}$

    $k' \leftarrow \hat{k}$

  end

  forall $\vec{p}^{\,*}_{(n_i,n_j),\hat{k}} \in P_{(n_i,n_j),k}$ **do**

    $\mathtt{ZS}(\vec{p}^{\,*}_{(n_i,n_j),\hat{k}}) \leftarrow \frac{\vec{p}^{\,*}_{(n_i,n_j),\hat{k}} - \mathtt{mean}(P_{(n_i,n_j),k})}{\mathtt{stddev}(P_{(n_i,n_j),k})}$

  end

  $P_{(n_i,n_j),k} \leftarrow \mathtt{ZS\_halve}(P_{(n_i,n_j),k})$

  $\dot{\vec{p}}_{(n_i,n_j),k} \leftarrow \mathtt{centroid}(P_{(n_i,n_j),k})$

  $\mathtt{PKT}^{i,j}_{\mathrm{rcv}} \xleftarrow{add} \left\langle k, \mathtt{RSSI}_{n_j,k}, \dot{\vec{v}}_{j,k}, \dot{\vec{v}}_{j,k-}, \dot{\vec{p}}_{(n_i,n_j),k} \right\rangle$

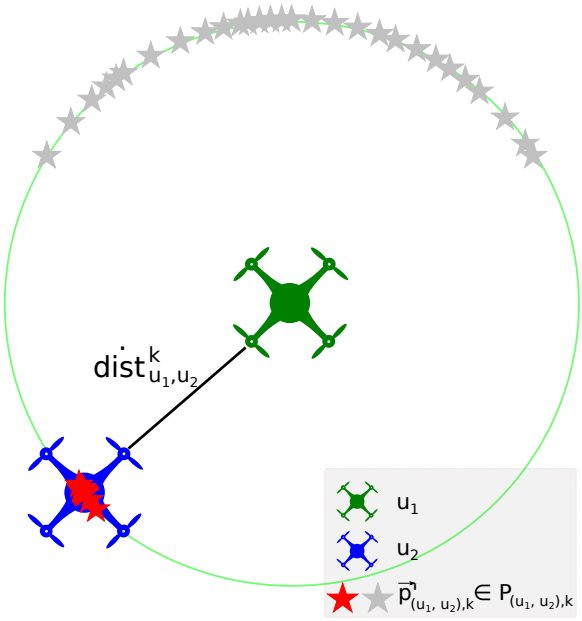


Fig. 5.5 Relative position estimation. The stars define the $P_{(i,j),k}$ set. The grey stars denote the points removed after the *Z-score* filter is applied.

At line 8, the term $\dot{\vec{p}}_{(n_i,n_i),\hat{k}}$ denotes the position estimation of n_i at time slot $t_{\hat{k}}$. The distance covered by node n_j from time slot $t_{\hat{k}}$ to t_k is derived at line 10, based on its average speed in the time interval (the $\dot{\vec{v}}_j$ variable at line 9). Knowing the original position, and the distance covered, a second estimation of the position of node n_j at time t_k ($\dot{\vec{p}}_{n_i \to n_j,\hat{k}}$) is derived in line 11 (the red UAV in Figure 5.4). Based on the positions produced so far ($\vec{p}_{(n_i,n_i),k}$ and $\dot{\vec{p}}_{n_i \to n_j)\hat{k}}$) and distances ($\dot{\text{dist}}^k(n_i,n_j)$ and $\dot{\text{dist}}^{\hat{k}}(n_i,n_j)$), the $\texttt{CI}(\vec{p1},d1,\vec{p2},d2)$ function (line 12) calculates the intersections between the circles with centers $p1$, $p2$ and radius $d1$, $d2$, respectively. Figure 5.4 shows the outputs of the circle intersection function, with two results being returned, denoted by the red UAV (correct estimation) and the grey one (wrong estimation). A single `HELLO` message does not allow discriminating the right solution, hence both the outputs are processed and inserted into the $P_{(n_i,n_j),k}$ set through the `Identity` function; this latter returns the arguments provided as inputs, and has been introduced only for ease of exposition, and more specifically to highlight the difference with the cooperative-based algorithm described in the next Section. Figure 5.5 depicts the solution set $P_{(n_i,n_j),k}$ computed by node n_i (green UAV) regarding the position of node n_j (blue UAV). Half of the estimations are clustered close to the real UAV position, while the other half is located on the circle of radius $\dot{\text{dist}}^k(n_i,n_j)$. In order to remove the outliers with respect to the main clusters, the *Z-score* index is computed at lines 17 - 19; based on such metric, half of the points are removed from the $P_{(n_i,n_j),k}$ set (function `ZS_halve` at line 20). This operation

is shown in Figure 5.5 where the grey points are removed from the solution set. Finally, at line 21, the centroid of $P_{(n_i,n_j),k}$ is returned as the relative position estimation of UAV u_j at time slot t_k.

Computational Complexity

The computational complexity CC of Algorithm 1 is determined by the main loop (lines 6-16 of Algorithm 1) where the set of possible solutions $P_{(n_i,n_j),k}$ is built. We assume that each node keeps only a limited number of received packets, defined by the system parameter $\mathrm{PKT}_{\max}$. It is easy to notice that $\mathrm{CC}(Algorithm1) = \mathcal{O}(\mathrm{PKT}_{\max})$.

5.3.2 Cooperative-based Neighbour estimator

The local algorithm described before might introduce some errors when pruning the solution set, as depicted in Figure 5.5. To this purpose, we propose an enhanced version of the algorithm, in which each node shares also its relative distance matrix with respect to its active neighbours: this is performed by extending the information contained in each $\texttt{HELLO}_{n_i,k}$ message as follows:

$$\left\langle n_i, \dot{\vec{v}}_{n_i,k}, \dot{\vec{v}}_{n_i,k^-}, \left[\left\langle n_j, \dot{\mathrm{dist}}^k(n_i,n_j) \right\rangle \dots \right] \right\rangle \forall n_j \in \mathrm{NB}_i \tag{5.24}$$

The cooperative-based neighbour estimator is mainly based on Algorithm 1, however it introduces a new method for the solution selection at each iteration. More in details, we replace the `Identity` function (line 13 in Algorithm 1) with the `ChooseCoop` function shown in Algorithm 2). Here, we consider the neighbours of n_i (i.e. the $n_q \in NB_i$ in line 3) which are also neighbours of n_j. A mean square positioning error for both the candidate solutions ($\vec{p}_1$ and $\vec{p}_2$) is computed by considering the distance estimation between n_q and n_k (contained in the `HELLO` message), and the estimated distance between n_i and n_q (computed locally by n_i). The solution associated with the lowest error (i.e. better fitting with the relative distance matrix) is returned and included in the set $P_{(n_i,n_j),k}$.

Figure 5.6 depicts the operations of the cooperative-based algorithm, by using the same notation of Figure 5.5: it is easy to notice that, even with more sparse position estimations, the algorithm is able to identify the correct UAV position within $P_{(n_i,n_j),k}$ thanks to the neighbours' relative distances information.

Algorithm 2: The COOPERATIVE algorithm

Input: $\mathrm{PKT}_{\mathrm{rcv}}^{i,j}$, $\mathrm{NB_q}, \forall n_q \in \mathrm{NB_i}$

Function `ChooseCoop`$(n_j, \vec{p}_1, \vec{p}_2)$**:**

  $p_1^{\mathrm{err}} \leftarrow 0$; $p_2^{\mathrm{err}} \leftarrow 0$;

  forall $n_q \in \mathrm{NB_i}, n_q \neq n_j$ **do**

    if $n_j \in \mathrm{NB_q}$ **then**

      $p_1^{\mathrm{err}} \leftarrow p_1^{\mathrm{err}} + (\left|\vec{p}_1 - \vec{p}_{(n_i,n_q),k}\right| - \mathrm{dist}^k(n_q,n_j))^2$

      $p_2^{\mathrm{err}} \leftarrow p_2^{\mathrm{err}} + (\left|\vec{p}_2 - \vec{p}_{(n_i,n_q),k}\right| - \mathrm{dist}^k(n_q,n_j))^2$

    end

  end

  if $p_1^{\mathrm{err}} < p_2^{\mathrm{err}}$ **then**

    return $\{\vec{p}_1\}$

  else

    return $\{\vec{p}_2\}$

  end

Function `OnPacketRCV`$(HELLO_{n_j,k})$**:**

  ...

  $f_{n_j,k} \leftarrow$ `ChooseCoop`$(n_j, \vec{p}'_{(n_i,n_j),\hat{k}}, \vec{p}''_{(n_i,n_j),\hat{k}})$

  ...

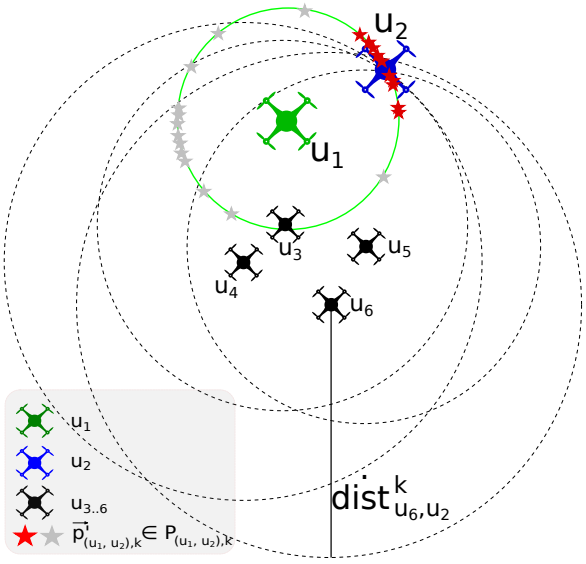


Fig. 5.6 Relative position estimation of UAV u_2 calculated by node u_1 using the cooperative-based algorithm.

Computational Complexity

The computational complexity of Algorithm 2 can be derived from the complexity of Algorithm 1. In addition to it, at each step of the mail loop, the `ChooseCoop` function is invoked; the latter iterates over the neighbour set in order to compute the localization errors for the candidate solutions $\vec{p}_1$ and $\vec{p}_2$. Hence, $\mathrm{CC}(Algorithm2) = \mathcal{O}(\mathrm{PKT}_{\max} \cdot |N|)$, where $|N| = N_U + N_H + N_R$.

5.3.3 Error Filter

Both the algorithms described before attempt to estimate the relative node position of node n_j by analyzing the list $\mathrm{PKT}_{\mathrm{rcv}}^{i,j}$ of received messages at node n_i. However, consecutive position estimations performed by the same node (n_i) on the same target (n_j) can incur into large variations/oscillations, due to noisy distance computations and low mobility conditions. For this reason, we included a filter mechanism that takes into account the history of the estimations produced so far and applies smoothing functions.

Algorithm 3: The Estimation Filter algorithm

Input: $\mathrm{PKT}_{\mathrm{rcv}}^{i,j}$
1 **Function** `EstimationFilter`$(\dot{\vec{p}}_{(n_i,n_j),k})$**:**
2 $\quad \mathrm{FI}_j \leftarrow \emptyset$
3 $\quad$ **forall** $\mathrm{pkt}_{\hat{k}}^{i,j} \in \mathrm{PKT}_{\mathrm{rcv}}^{i,j}$ **do**
4 $\quad\quad \dot{\vec{\mathrm{est}}}_{\hat{k}} \leftarrow \dot{\vec{p}}_{(n_i,n_j),\hat{k}} + \dot{\vec{v}}_{n_j}(\hat{k},k) \cdot (k-\hat{k}) \cdot t_{\mathrm{slot}}$
5 $\quad\quad w_{\hat{k}} \leftarrow f_{\mathrm{filter}}^{(k-\hat{k})} \cdot 1/e^{\texttt{dispersion}(P_{(n_i,n_j),\hat{k}})^2}$
6 $\quad\quad \mathrm{FI}_j \leftarrow \left\langle \dot{\vec{\mathrm{est}}}_{\hat{k}}, w_{\hat{k}} \right\rangle$
7 $\quad$ **end**
8 $\quad \dot{\vec{p}}^{\mathrm{filt}} \leftarrow \texttt{WeightedAverage}(\mathrm{FI}_j)$
9 $\quad p_{\mathrm{dist}} \leftarrow |\dot{\vec{p}}^{\mathrm{filt}} - \dot{\vec{p}}_{(n_i,n_j),k}|$
10 $\quad \max_{\mathrm{dist}} \leftarrow 2 \cdot \dot{\mathrm{dist}}^k(n_i,n_j)$
11 $\quad$ **if** $\max_{\mathrm{dist}} > p_{\mathrm{dist}}$ **then**
12 $\quad\quad f_{\mathrm{factor}} \leftarrow p_{\mathrm{dist}}^2 / \left(p_{\mathrm{dist}}^2 + (\max_{\mathrm{dist}} - p_{\mathrm{dist}})^2\right)$;
13 $\quad$ **else**
14 $\quad\quad f_{\mathrm{factor}} \leftarrow 1$
15 $\quad$ **end**
16 $\quad$ **return** $\dot{\vec{p}}^{\mathrm{filt}} \cdot f_{\mathrm{factor}} + \dot{\vec{p}}_{(n_i,n_j),k} \cdot (1 - f_{\mathrm{factor}})$

Algorithm 3 shows the pseudocode of the proposed error filtering mechanism. Within the main loop (lines 3 - 7), we derive $\dot{\vec{\mathrm{est}}}_{\hat{k}}$, i.e. the estimated position of node $n_j \in N$ based on the previous estimation calculated at time slot $t_{\hat{k}}$ and on the average velocity till time

slot t_k. A weight coefficient $w_{\hat{k}}$ quantifies the accuracy of each estimation, by considering two factors (line 5): (*i*) the temporal freshness of the information, since positioning errors may accumulate over time and hence too old estimations can be largely inaccurate[1]; (*ii*) the trustworthiness of the estimation, reflected by an *index of dispersion* (i.e. the `dispersion` function) that is defined as $\texttt{dispersion} = \sigma^2/\mu$. The weighted average of the estimations produced so far, i.e. $\dot{\vec{p}}^{\text{filt}}$, is produced at line 8. Finally, the position of node n_j is built as combination between the original estimation $\dot{\vec{p}}_{(n_i,n_j),k}$ (i.e. the output of the previous algorithms) and the weighted historic average, $\dot{\vec{p}}^{\text{filt}}$. Again, the the factor $f_{\text{factor}} \in]0..1[$ (lines 11 - 15) works as a weighting coefficient, and it gives more trust to the historic average or to the original estimation based on the distance between $\dot{\vec{p}}_{(n_i,n_j),k}$ and $\dot{\vec{p}}^{\text{filt}}$ (we use the estimated distance as reference max distance, see line 10).

Computational Complexity

Similarly to Algorithm 1, the computational complexity of Algorithm 3 is dominated by the loop over the received packets, hence: $\text{CC}(Algorithm3) = \mathcal{O}(\text{PKT}_{\max})$.

5.3.4 PSO optimizer

Finally, we further enhance the position estimations produced so far by means of a Particle Swarm Optimization (PSO). Since the latter exploits the knowledge of each node n_i neighbourhood NB_i, it can be used only when the cooperative technique (Section 5.3.2) is enabled. The PSO is a heuristic computational technique which explores a solution space for the optimum search, by means of a set of candidate solutions named *particles*. At each iteration, the candidate solutions are updated by adjusting the particle's position and velocity with respect to a goal function.
In our scenario, the PSO particles correspond to the relative positions of each node $n_j \in \text{NB}_\text{i}$. The PSO technique -described in Algorithm 4- is executed by node n_i each t_{PSO} time slots, and the goal is to minimize an error/loss positioning function introduced later in this Section.

Let PA be the set of particles and pa_w the w-th particle, with $w \leq N_{\text{pa}}$. Moreover, let $\text{pa}^{\text{pos}}_{w,j}$ be the candidate relative position of node $n_j \in \text{NB}_\text{i}$ with respect to node n_i belonging to the w-th particle, and $\text{pa}^{\text{vel}}_{w,j}$ its velocity. Here, N_{pa} is a system variable and defines the number of used particles. The structure of each particle is shown in Figure 5.7. In the initialization phase (lines 2-12 of Algorithm 4), all the particles are randomly initialized, except one which corresponds to the output of the cooperative-based algorithm (line 5). The

[1] $f_{\text{filter}} \in [0..1]$ is a system parameter, powered to the freshness of the information given by the time interval $t_k - t_{\hat{k}}$.

Algorithm 4: The PSO optimization algorithm

Input: $\mathrm{NB_i}$; $\mathrm{NB_j}, \forall n_j \in \mathrm{NB_i}$

1 **Function** `ExecPSO()`**:**
2 **forall** $\mathrm{pa}_w \in \mathrm{PA}$ **do**
3 **forall** $n_j \in \mathrm{NB_i}$ **do**
4 **if** $w = 1$ **then**
5 $\mathrm{pa}^{\mathrm{pos}}_{w,j} \leftarrow \vec{p}_{(n_i,n_j),k}$
6 **else**
7 $\mathrm{pa}^{\mathrm{pos}}_{w,j} \leftarrow \left\langle \mathtt{rnd}(x^j_{min}, x^j_{max}), \mathtt{rnd}(y^j_{min}, y^j_{max}) \right\rangle$
8 **end**
9 $\mathrm{pa}^{\mathrm{vel}}_{w,j} \leftarrow \mathtt{rnd}(v_{min}, v_{max})$
10 **end**
11 $\mathrm{pa}_{\mathrm{loc}_w} \leftarrow \mathrm{pa}_w$
12 **end**
13 **for** $\mathrm{ite} = 1; \mathrm{ite} \leq \mathrm{ite_{PSO}}; \mathrm{ite}{+}{+}$ **do**
14 **forall** $\mathrm{pa}_w \in \mathrm{PA}$ **do**
15 **forall** $n_j \in \mathrm{NB_i}$ **do**
16 $\mathrm{pa}^{\mathrm{vel}}_{w,j} \leftarrow \omega \cdot \mathrm{pa}^{\mathrm{vel}}_{w,j} +$
17 $+ c_{\mathrm{loc}} \cdot \mathtt{rnd}(0,1) \cdot \left(\mathrm{pa}^{\mathrm{pos}}_{\mathrm{loc}_w,j} - \mathrm{pa}^{\mathrm{pos}}_{w,j} \right) +$
18 $+ c_{\mathrm{glob}} \cdot \mathtt{rnd}(0,1) \cdot \left(\mathrm{pa}^{\mathrm{pos}}_{\mathrm{glob}_w,j} - \mathrm{pa}^{\mathrm{pos}}_{w,j} \right)$
19 $\mathrm{pa}^{\mathrm{pos}}_{w,j} \leftarrow \mathrm{pa}^{\mathrm{pos}}_{w,j} + \mathrm{pa}^{\mathrm{vel}}_{w,j}$
20 $\mathtt{checkIfOutside}(\mathrm{pa}^{\mathrm{pos}}_{w,j})$
21 **end**
22 $\mathrm{pa}_{\mathrm{loc}_w} \leftarrow \mathrm{argmin}_{\mathrm{pa}_z \in \{\mathrm{pa}_w, \mathrm{pa}_{\mathrm{loc}_w}\}} \mathscr{L}(\mathrm{pa}_z)$
23 **end**
24 $\mathrm{PA_{loc}} \leftarrow \bigcup_{\mathrm{pa}_w \in \mathrm{PA}} \mathrm{pa}_{\mathrm{loc}_w}$
25 $\mathrm{pa_{glob}} \leftarrow \mathrm{argmin}_{\mathrm{pa}_z \in \mathrm{PA_{loc}} \cup \{\mathrm{pa_{glob}}\}} \mathscr{L}(\mathrm{pa}_z)$
26 **end**
27 **forall** $n_j \in \mathrm{NB_i}$ **do**
28 $\vec{p}_{(n_i,n_j),k} \leftarrow \mathrm{pa}^{\mathrm{pos}}_{\mathrm{glob},j}$
29 **end**

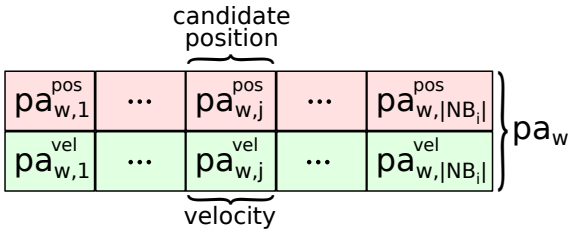


Fig. 5.7 The w-th particle definition in the PSO algorithm.

search space of the particles is defined by the variables x^j_{min}, x^j_{max}, y^j_{min}, and y^j_{max}. We define $x^j_{max} = y^j_{max} = 2 \cdot \dot{\text{dist}}^k(n_i, n_j)$ and $x^j_{min} = y^j_{min} = -2 \cdot \dot{\text{dist}}^k(n_i, n_j)$, while the `rnd` function returns a random number within the range passed as input. Similarly, the particle's velocity is randomly initialized (line 9) within the range v_{min} and v_{max}. The algorithm keeps track of the the best local candidate solution (visited by particle pa_{loc_w}) and also of the best global candidate solution (visited by particle pa_{glob}). The space exploration is performed within the main loop which is executed for a number of ite_{PSO} iterations (lines 13-26). At each iteration, the particle's position and velocity are updated (lines 15-21) and then, the best and local candidates are also updated in line 22 and line 25, respectively. Finally, the pa_{glob} is returned as the final result (line 28). The velocity update rule is defined as in [79]: ω indicates the inertia weight factor; c_{loc} and c_{glob} quantifies the attraction force toward the local and global best values, respectively. The overall rationale is that each particle has its own velocity and hence its inertial force, but it is still attracted by the local and global optimal solutions discovered so far. The randomness factor enforces the space exploration; to this purpose, the `checkIfOutside` function ensures that the candidate positions lie within the search space defined by x^j_{min}, x^j_{max}, y^j_{min}, and y^j_{max}.

The PSO loss is modeled by the $\mathscr{L}(\text{pa}_z)$ function, which is executed on node $n_i \in N$ at time slot t_k and is defined as follows:

$$\begin{aligned}\mathscr{L}(\text{pa}_z) = \sum_{n_j \in \text{NB}_i} \Big(& |\text{pa}^{\text{pos}}_{z,j} - \dot{\vec{p}}_{(n_i,n_j),k}|^2 \\ & + \big(|\text{pa}^{\text{pos}}_{z,j}| - \dot{\text{dist}}^k(n_i,n_j)\big)^2 \\ & + \sum_{\substack{n_q \in \text{NB}_i \\ [n_q \neq n_j, \\ n_j \in \text{NB}_q]}} \big(|\text{pa}^{\text{pos}}_{z,q} - \text{pa}^{\text{pos}}_{z,j}| - \dot{\text{dist}}^k(n_q,n_j)\big)^2 \Big)\end{aligned} \tag{5.25}$$

Here, the second row computes the positioning error of node n_j with respect to the local distance estimation $\dot{\text{dist}}^k(n_i, n_j)$, whereas the third line computes the positioning error with respect to the estimations produced by the other neighbours of n_i. It is worth highlighting

that the optimal solutions might be infinite since the loss function takes into account only the nodes' distances; hence, given an optimal solution, a new one can be produced by a simple rotation. To avoid the problem, we used as fixed anchor the previous relative position estimation of node n_j (described in the first row of Equation (5.25)).

Computational Complexity

The computational complexity of the PSO technique is dominated by the main loop over the particles. i.e. by lines 13-26 of Algorithm 4. Here, three additional, consecutive loops are executed for the particles' updates: the first is bound by the number of iterations ite_{PSO}; the second goes over all the particles; the third visits all the node neighbours, i.e. $\mathcal{O}(|N|)$. Hence, we can state that $\text{CC}(Algorithm4) = \mathcal{O}(\text{ite}_{\text{PSO}} \cdot N_{\text{pa}} \cdot |N|)$.

5.4 Performance Evaluation

In this Section, we evaluate the system performance of the ELAPSE framework by a twofold evaluation. First, in Section 5.1, we investigate the swarm creation and management for the aerial coverage of large-scale, disaster areas by means of extensive OMNeT++ simulations. Then, we focus the attention on selected components of our framework (e.g. the QoS support), and demonstrate their effectiveness through a small-case testbed composed of autonomous ground robots.

5.4.1 OMNeT++ Simulations

We modelled the scenario characteristics and the wireless communications on the AtA and AtG links in OMNeT++, by creating new modules for the virtual-spring based swarm mobility algorithms and the positioning techniques. Similarly, we modeled the position of the Help Requesters via a Markov-Gaussian mobility model (to simulate the pedestrian mobility), and of mobility of each Rescue Personnel via a Random-direction mobility model (to simulate a search mobility around the scenario). Unless stated otherwise, we used the following parameters: $N_U = 12$, $N_H = 20$, $N_R = 6$, $f_a = 10$ m, $\text{dist}_{\text{min}} = 10$ m, $k(MtM) = 120$, $k(MtH) = 100$, $KT = 180$, $t_{\text{dec}} = 3$ s, $t_{\text{slot}} = 0\,1$ s, $t_{\text{broadcast}} = 5$, $\alpha_{AtA} = 2$, $\alpha_{AtG} = 2.6$, $\sigma_d = 5m$, $t_{\text{timeout}} = 50$, $f_{\text{factor}} = 0.85$, $t_{\text{PSO}} = 20$, $N_{\text{PA}} = 20$, $ite_{\text{PSO}} = 120$, $\omega = 0.1$, $c_{\text{loc}} = c_{\text{glob}} = 1.25$. In the evaluation, we keep uniform the value of $LB^k_{\text{req}}(n_i, n_j) = LB_{\text{req}} =$ 12 dB.

Relative Position Estimation

The first analysis investigates the performance of the GPS-free positioning techniques proposed in Section 5.3, and more specifically the average accuracy of each node in estimating the relative positions of its neighbours. To this aim, we compare the two methods described in Section 5.3, i.e. the *local-based* and *cooperative-based* algorithms.

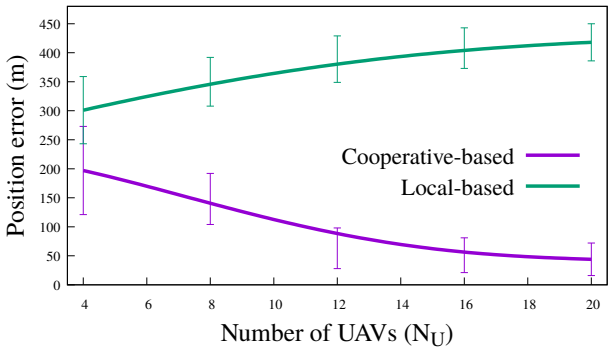


Fig. 5.8 Average position estimation error when varying the number of UAVs N_U.

Figure 5.8 shows the average neighbour position error when varying the number of UAVs composing the swarm. As we can notice from the plot, the two methods follow different trends, i.e. the average error increases with the number of UAVs for the *local-based* algorithm while it decreases for the *cooperative-based* one. For the *local-based* solution, the trend can be explained by the error accumulation; indeed, each local neighbour estimation introduces some positioning errors, which impact the computation of the resultant spring force in Equation 5.17 and hence the (wrong) positioning of the node. The spring direction error will clearly increase when considering more forces, i.e. when increasing the UAV density. Vice versa the *cooperative-based* algorithm exploits the neighborhood information in order to select the next position of each neighbour, between the speed-based and a RSS-based estimations (the `ChooseCoop` function of Algorithm 2). The higher is the number of neighbors, the higher becomes the probability to remove potential outliers. Figure 5.8 shows that the error is in the order of a tens of meters, which is still quite high in absolute way, however tolerable in relative way when compared with the average node distance which is in the order of 300 meters.

Figure 5.9 highlights the impact of the different building blocks of Figure 5.3 on the final position error of the *cooperative-based* algorithm. We can see that the error filter and the optimizer blocks introduce higher performance gains with low number of UAVs (e.g. $N_U \leq 10$), since the neighbour estimation module has more uncertainty regarding the

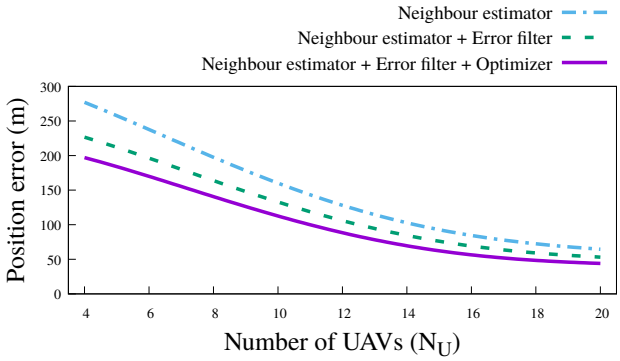


Fig. 5.9 The position error after each block of the Position Estimation algorithm when varying the number of UAVs N_U.

scenario. For the same reason, the impact of the two blocks is reduced when increasing the number of UAVs.

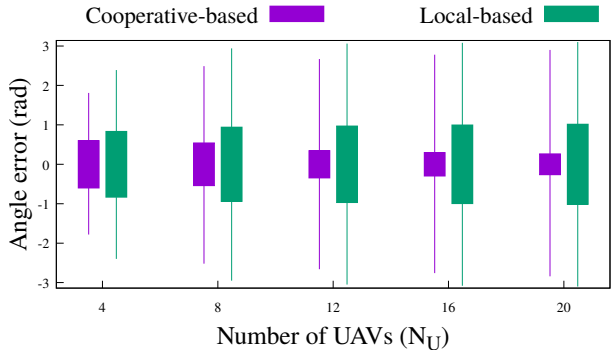


Fig. 5.10 Relative angle estimation error over the number of UAVs N_U.

The same trend of Figure 5.8 is confirmed by Figure 5.10 which shows the angle/direction error of the positioning techniques. Again, the *cooperative-based* algorithm overcomes the *local-based* and it improves its performance with increasing densities of UAVs.

Finally, we analyze the capability of the proposed algorithms to cope with noisy distance estimations; to this purpose, in Section 5.3, we introduced the σ_d parameter which models the error on the RSS-based distance estimator. Figure 5.11 shows the average position errors when varying the σ_d values on the x-axis. As expected, both the *single-based* and the *cooperative-based* algorithm degrade their performance when increasing the error on the input. However, while the error for the *single-based* almost triple from σ_d=0 to σ_d=20, the increase is limited to few meters for the *cooperative-based* algorithm. This result confirms the robustness of the proposed technique also over noisy channels.

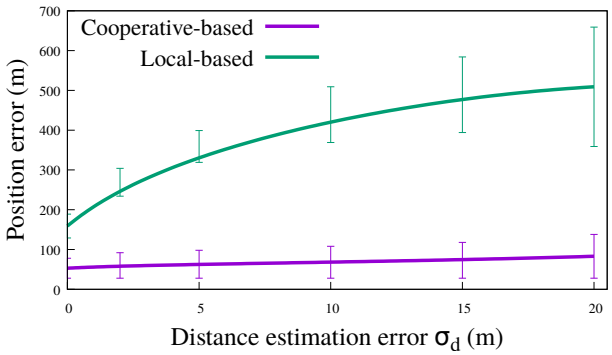


Fig. 5.11 Position estimation error when varying the distance estimation noise σ_d.

Scenario Coverage

In this analysis, we investigate the ability of the swarm mobility model of Section 5.2 to provide multi-hop connectivity among the MGNs, and the impact caused by the positioning algorithms on the overall coverage. It is worth reminding that the connections among ground/aerial nodes, and the mobility of the UAVs are both governed by the abstraction of virtual spring forces. Obviously, the higher is the accuracy in estimating the nodes' positions, the higher is the ability to create a connected aerial mesh network and to discover the ground nodes.

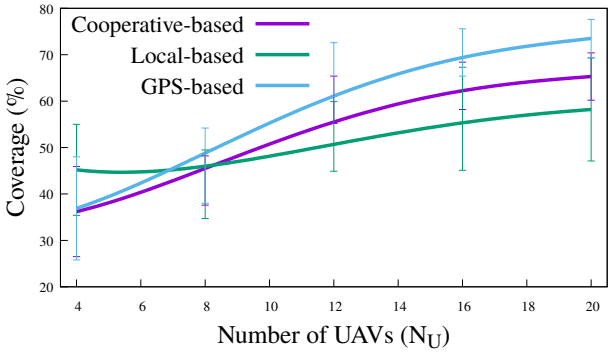


Fig. 5.12 Percentage of covered help requesters when varying the number of UAVs N_U.

Figure 5.12 shows the percentage of the help requesters covered by at least one UAV. As in the previous Section, we considered the *local-based* and *cooperative-based* algorithms, and compared them against a reference GPS-based solution where all the UAVs are equipped with a geo-localization device (also, assumed error-free). Clearly, the latter constitutes an upper bound on the system performance. It is easy to notice that the coverage percentage increases with the number of UAVs and that the *cooperative-based* algorithm approaches the

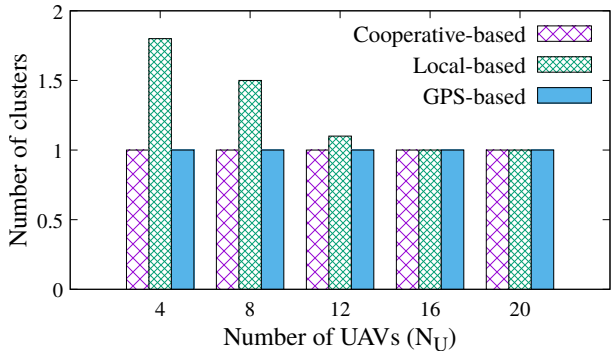


Fig. 5.13 Number of UAVs cluster when varying the number of UAVs N_U.

GPS-based one. Vice versa, the *local-based* seems to maximize the coverage when $N_U \leq 6$, while its performance degrades quickly for larger values of UAVs due to the impact of the positioning errors on the configuration of the virtual springs. A better insight on this analysis is provided by Figure 5.13 which shows the average number of UAV isolated clusters created during the simulation. The aerial mesh connectivity is a constraint of our problem (Equation 5.5), and it is always met by the GPS-based and *cooperative-based* algorithms. Vice versa, for the *local-based* algorithm, isolated clusters might occur due to AtA link breakages when the number of UAVs -and hence the number of virtual forces acting on each node- is low. Clearly, multiple, independent aerial mesh networks can cover larger areas than a single swarm; this explains the higher performance of the *local-based* algorithm in Figure 5.12 for low UAV density values. At the same time, isolated clusters do not allow coordinated emergency responses that involve all the ground nodes connected, and for this reason they are considered as a goal of the ELAPSE framework.

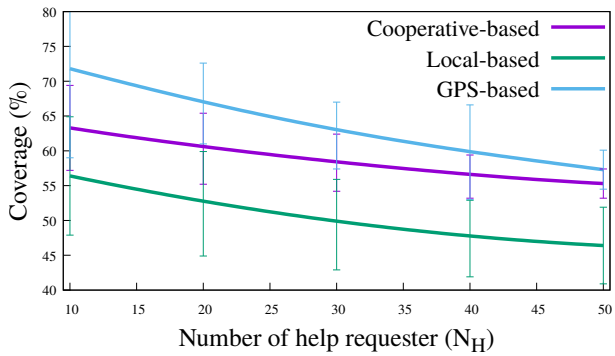


Fig. 5.14 Percentage of covered help requesters when varying the number of help requester N_H.

Figure 5.14 shows the coverage percentage when we vary the number of help requesters N_H while keeping constant the number of UAVs ($N_U = 12$). Also in this case we can notice that the *cooperative-based* algorithm performs very similar to the GPS-based system, and that the performance gap reduces when increasing the number of N_H. This result confirms the ability of the our solution to maximally exploit the information exchanged by an increasing number of cooperative nodes.

Aerial Network Deployment

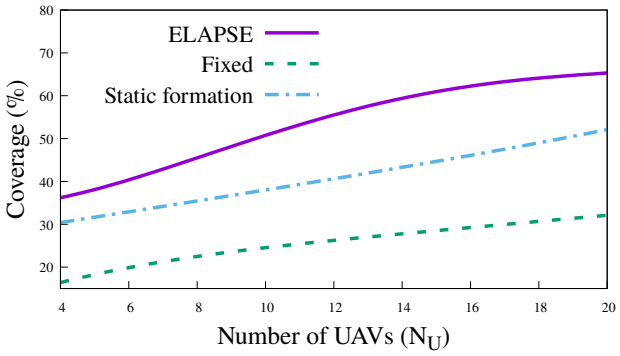


Fig. 5.15 Percentage of covered help requesters when varying the number of UAVs N_U.

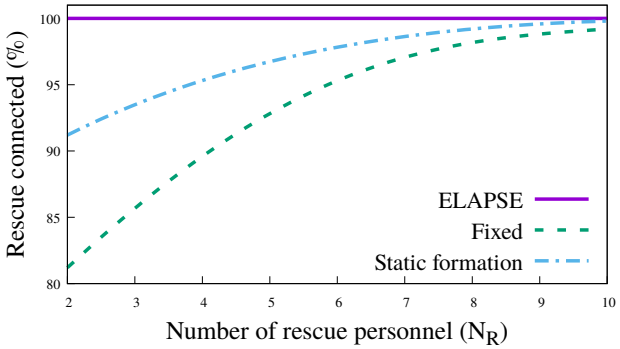


Fig. 5.16 Percentage of time in which at least one rescue personnel is covered, when varying the number of rescue personnel N_R in the scenario.

The last analysis investigates the QoS support of the aerial mesh network, and the overall multi-hop connectivity between the help requesters and the rescue teams, which is the final goal of the ELAPSE deployment in emergency scenarios.
To this purpose, we compare the ELAPSE framework against two alternative solutions for aerial mesh deployments: (*i*) a *Fixed* deployment, where the UAVs are placed according to a

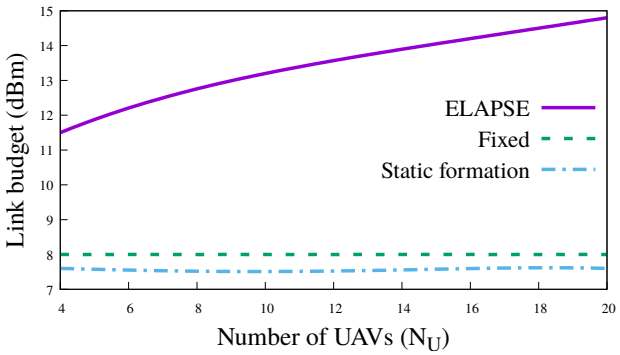


Fig. 5.17 Average link-budget of the wireless links when varying the number of UAVs N_U.

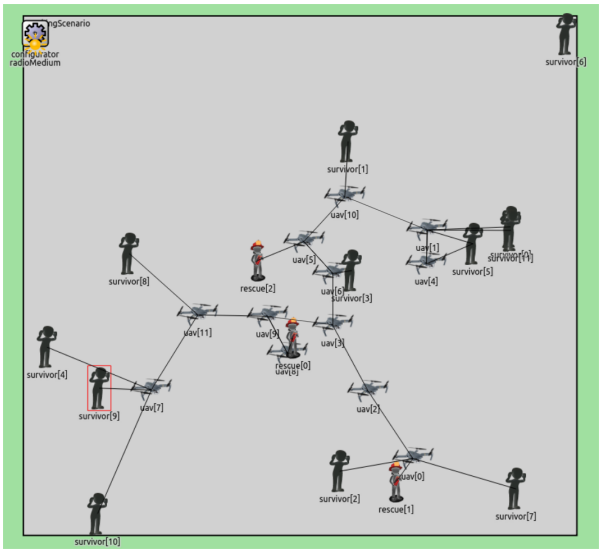


Fig. 5.18 A screenshot of an OMNeT++ simulation, showing the ability of the ELAPSE framework to adapt its deployment to the ground mobility of the MGNs.

connected grid formation at the center of the scenario; (*ii*) *Static formation* where again a specific grid formation is considered, however not anchored to static positions, i.e. the UAVs can continuously move over the scenario. In both cases, the grid formation is designed in order to guarantee a link budget value equal to LB_{ref} between each couple of neighbours UAVs.

Figure 5.15 compares the three deployment methods in terms of coverage percentage of the help requesters. We can notice that the ELAPSE framework greatly overcomes the other two methods thanks to its flexibility, i.e. the possibility to dynamically adapt the UAV formation (also assuming irregular shapes) based on the current location of the ground nodes; vice versa the *Fixed* approach provides the worst performance due the limited exploration of the scenario. A visual evidence of the self-organization capabilities of the ELAPSE framework is

provided by Figure 5.18, which shows a screenshot of the OMNeT++ simulation; we can notice the irregular formation of the self-configuring aerial mesh which is able to provide global coverage of the MGNs.
Figure 5.16 further analyzes the dual coverage metric, this time in terms of rescue personnel connected to the UAVs. More specifically, we plot on the y axis the percentage of time-slots in which at least one rescue personnel is connected to the aerial mesh, while on the x axis we vary the number of rescue personnel available within the scenario. We notice that the constraint is not always satisfied over time by the *Fixed* and the *Static formation* methods, while it is always satisfied by the ELAPSE framework through the MtG virtual spring mechanism. Considering this result in conjunction with Figures 5.13 and 5.14, we can conclude that the ELAPSE framework is able to support rescue operations in an effective manner, by guaranteeing end-to-end connectivity among isolated help requesters and rescue personnel.
Finally, in Figure 5.17 we analyze the average link budget available on each communication link (here $LB_{\text{ref}} = 12$ dB). We can notice that both the *Fixed* and the *Static formation* methods do not guarantee the QoS on the communication links. Indeed, despite the fact that -by construction- the UAVs are placed in order to meet the requested link budget, i.e. LB_{ref}, the AtG links might experience much lower values. Vice versa, the ELAPSE framework, by taking into account the LB_{ref} value inside the virtual spring Equation, is able to guarantee uniform link qualities on both AtA and AtG links, and to meet the QoS constraint on almost all the UAV configurations.

5.4.2 Ground vehicle Test-Bed

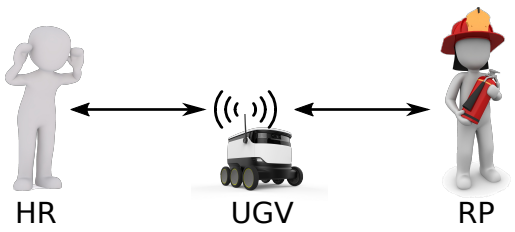


Fig. 5.19 The test-bed scenario with the ground robot.

We further characterized the performance of selected components of the ELAPSE framework through a small-case testbed. More in details, we consider the experimental setup depicted in Figure 5.19. Here we consider one help requester (on the left), one rescuer (on the right) and one autonomous Unmanned Ground Vehicle (UGV), which must move within the scenario, discover the nodes, and control its own position in order to provide QoS-aware

end-to-end connectivity among the two end-points. To this purpose, the MGN nodes are constituted by two Bluetooth Low Energy (BLE) devices, periodically broadcasting HELLO messages with the payload defined in Section 5.20. Beside the wireless link differences in comparison with the aerial vehicles, the UGVs will be able to test the core of the ELAPSE framework, i.e. the swarm mobility algorithm and the positioning technique.

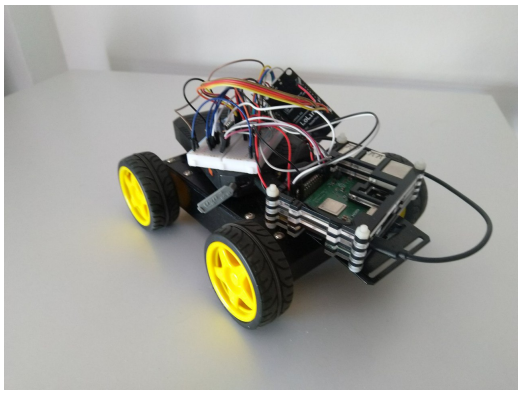


Fig. 5.20 The UGV used in our test-bed

The UGV is the ground robot depicted in Figure 5.20, equipped with the hardware below: (*i*) a Raspberry PI (model 3B+) board, which embeds the overall UGV controller, including the spring algorithm, the positioning technique (in this case, the UGV attempts to estimate the position of the two BLE devices), the sensor data acquisition and the wireless communications; (*ii*) an IMU, model GY-88, including accelerometer, gyroscope and magnetometer sensors used for speed computation and direction estimation; (*iii*) a NodeMCU microcontroller, used for the sensor data acquisition; (iv) two 2.4 GHz radio interfaces (embedded within the Raspberry board) for the wireless communications, i.e. BLE to communicate with the two MGNs and Wi-Fi to communicate with a laptop used for statistics collection. We performed the tests on an outdoor scenario, with a distance of 8.5 meters between the two MGNs, assumed static. The software has been implemented using the Johnny-Five[2] framework, which communicates with NodeMCU via the Firmata protocol. The software components have been dockerized and deployed on the BalenaCloud[3], a container-based platform for IoT applications.

First, we estimated the path-loss model, which is used to calibrate the RSS-based distance estimation (from the BLE signal), and more specifically the α_{GtG} and κ_{GtG} parameters of Section 5.1.3. We highlight that the goal here is slightly different from the original problem since we are not considering aerial network deployments, hence the path loss refers to

[2]https://github.com/rwaldron/johnny-five
[3]https://www.balena.io/cloud/

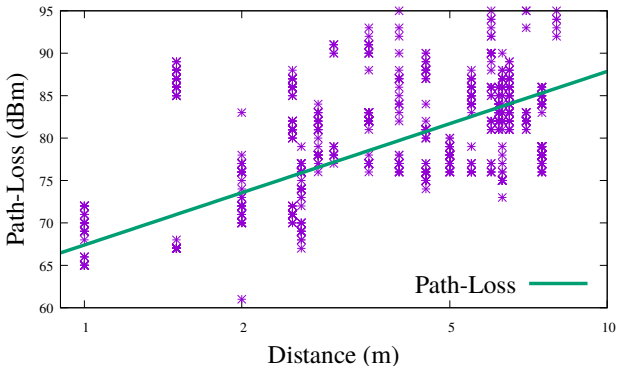
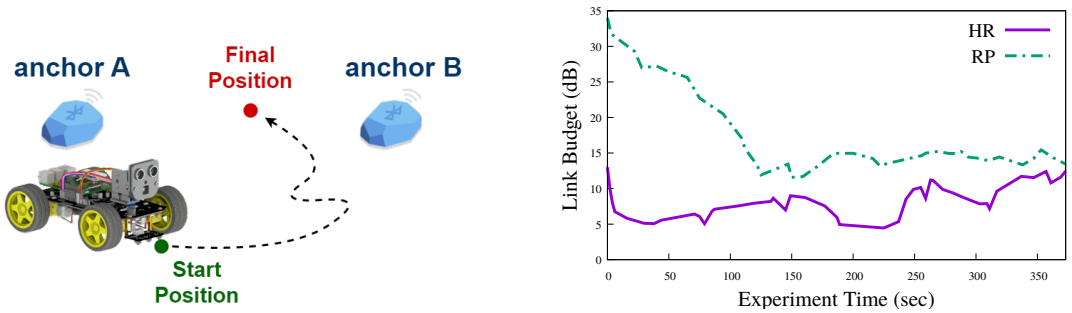


Fig. 5.21 The Path-Loss model over distance

Ground-to-Ground (GtG) links. We collected a consistent set of BLE samples at different real distances, denoted as points in Figure 5.21. The path-loss estimation, i.e. the fitting line in the curve, is computed as follows:

$$PL(d) = 10 \cdot 2.046 \cdot \log_{10}(d) + 64.4 \tag{5.26}$$



(a) (b)

Fig. 5.22 In Figure 5.22a the UGV reaching the best spot between the two BLE anchors A and B and Figure 5.22b shows the measured Link Budget (LB) values over time on the two GtG links

In Figure 5.22b we demonstrate the QoS support offered by the virtual spring mechanism described in Section 5.2. Specifically, we consider a situation where LB_{ref}=15 dB. On the graph we depict the LB values on the wireless links, i.e. from the help requester to the UGV (purple line) and from the rescue personnel to the UGV (green line). The UGV is initially placed at a random position of the scenario, but closer to the rescuer node. It is easy to notice that the UGV progressively adjusts its 2D position over time in order to balance the LB on both links; this corresponds more or less to the central position between the MGNs. After

300 seconds, both the LB values converge to the requested threshold, and hence the UGV achieves a stable position.

Number of received beacons	Error ratio
5	1.78
15	1.73
20	1.67
25	1.08

Table 5.1 Positioning Error Ratio with respect to the number of `HELLO` messages received by the UGV.

We conclude the analysis by investigating the relationship between the number of `HELLO` messages received by the UGV and the accuracy of the neighbour estimator. We consider the *cooperative-based* algorithm, executed by UGV in order to estimate the relative position of the each MGN. In the table below, the error is computed as the ratio between the average positioning error (i.e. the difference between the real position of a MGN and the estimation computed by the UGV) and the actual distance among the nodes. It is easy to notice that the error ratio decreases considerably with the number of received updates, i.e. the more the UGV moves within the scenario, the higher is the accuracy of the neighbors' knowledge acquired by the UGV.

Chapter 6

Uhura: enabling hybrid mesh networking

Chapter 3 identified the third objective of this thesis as investigating hybrid scenarios with air-to-ground communication links between the ground and aerial WMSN meshes. In Chapters 4 and 5, we presented our contributions to addressing this objective, including the development of ground and aerial WMSN solutions, as well as a novel swarm mobility algorithm called ECLIPSE. In this chapter, we investigate the integration of the ground and aerial network segments previously described using the Uhura, a novel software framework that facilitates communication within an aerial swarm and between the swarm and ground devices.

As discussed in Chapter 3, the Uhura framework was designed and implemented to meet the four requirements of this thesis. Specifically, the Uhura Adapter and Uhura Core modules meet the *Device Heterogeneity* and *Adaptive QoS Management* requirements, respectively, by enabling multiple stack M2M communication in an agnostic way. Additionally, we describe two relevant modules of the Uhura core architecture in this chapter. The Service Discovery System meets the *Scalability* requirement by enabling efficient discovery and utilization of services within the hybrid network, while the Autonomic Faulty Node Replacement module meets the *Reliability* requirement by enabling the detection and replacement of faulty nodes in the network.

Overall, this chapter presents our work on integrating the ground and aerial WMSN segments using the Uhura framework, including our implementation details, evaluation of performance, and potential applications.

Introduction

Thanks to the recent advances in technology and to the decreasing cost of hardware solutions, in the last decade, the use of robotic swarms increased exponentially. Swarm behaviors offer scalability and robustness to failure which is often ensured by decentralized and distributed

design methodologies. However, such approaches usually rely on local interactions that have somehow to be assured by the underlying communication technologies installed on the platform [80].

For instance, drone shows are the most well-known displays of swarming technologies among the many applications. These are also the most impressive, with 1,824 unmanned aerial vehicles at the Tokyo Olympics and a record-breaking show that involved up to 3,281 in a show set up by a famous luxury car retailer [81]. Despite the impressive numbers, as reported in Coppola et al. [82], these demonstrations lack of the decentralized and distributed properties of swarm robotics approaches and, in most cases, rely on traditional communication technologies. In particular, broadcast mechanisms have often been employed in the more classic swarm robotics to cope with the devices' limited computational/networking capabilities [83]. Although many different solutions can be found in the literature, they are mainly constrained to only one specific wireless technology: this is the case of simple flooding mechanisms [84], Geo-Aware flooding approaches [85] and synchronous flooding over IEEE 802.15 standard [86].

However, the trend is changing due to the new possibilities and challenges posed by the growth of the Internet of Things (IoT). On the one side, new services and applications of swarm robotics demand the possibility to communicate with ground IoT devices, hence imposing the need for different wireless technologies on the ground-to-grand ground-to-air and air-to-air links. On the other side, several Machine-to-Machine (M2M) communications stacks are nowadays available on the market [7] with different Quality of Service (QoS) support in terms of range, bandwidth, and energy consumption; this may pose formidable challenges for the technology integration but may also constitute a boosting factor for robotic mesh development.

In this work, we move away from the single technology scenario by investigating heterogeneous robotic networks composed of swarms of Unmanned Aerial Networks (UAVs) or Ground vehicles (UGVs) interacting with each other and with ground IoT devices. To this aim, we present *Uhura*, a novel software framework that enables communication within the swarm and from the swarm to the IoT in an M2M agnostic way. Indeed, a *Uhura* node can be equipped with multiple M2M stacks and can manage them through the abstraction of *Uhura Adapter*. In addition, the *Uhura Core* module allows dynamic selection of the best adapter to use on each link of the swarm based on the QoS requirements of the application. The chapter is structured as follows. Section 6.1 introduces the architecture and main components of the *Uhura* framework. Section 6.3 provides few details about the implementation. Section 6.2 illustrates three generic use-cases of the *Uhura* framework. Experimental setup and

results are presented in Section 6.4. Finally, Section 6.6 shows a Service Discovery System implemented over Uhura Framework.

6.1 Software Architecture

Communication technologies play a key role in the deployment of swarm robotic networks. Although most of the existing testbeds reported in the literature employ the WiFi either in single-hop centralized or multi-hop mesh modes for performance and easiness of setup reasons [80], an incredible number of Machine to Machine (M2M) communication technologies are available on the market, also as a consequence of the explosion of the Internet of Things (IoT) devices [7]. Integrating such low-power devices with the robotic swarms may be challenging and require using different technologies on the sensor-to-robot and robot-to-robot links. Similarly, multi-technology swarm robotic networks are gaining progressive interest due to the possibility of supporting Quality of Service (QoS) requirements by introducing per-link technology differentiation and/or completely separating the data and control planes [87]. Here the *Uhura* framework comes to play: it provides functionalities to deploy integrated robotic systems composed of UAV, UGV, and ground sensors in a communication-agnostic way. More specifically, the framework includes the concept of *Uhura Adapter* as a software module aimed to enable message exchange with a specific M2M technology. Our platform is easily extensible, and new modules can be easily integrated in order to support other technologies by implementing a common API. In addition, the *Uhura* framework allows monitoring of the network performance of each active adapter towards a neighbor node. It dynamically selects the one to use for message transmission/reception, so that (*i*) per-link QoS requirements are continuously met and/or (*ii*) robustness of communication is increased by switching from one adapter to another in case of wireless link failure.

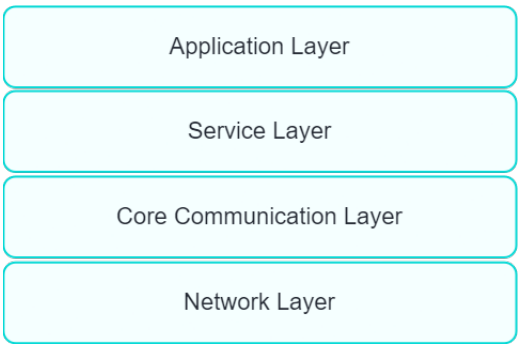


Fig. 6.1 *Uhura* high level layered architecture

The high-level layered architecture of Uhura is depicted in 6.1; it shows four layers starting from the *Network layer* that includes the multiple instances of *Uhura Adapter* running for each M2M technology, then *Core Communication layer* where *Uhura Core* taking his place and eventually new core modules can be added in the future. The *Service Layer* composed of two components, *Discovery* and *Gateway*, largely described in Section 6.6 that enables, as names suggest, a service deploy and discovery system for Uhura; lastly, the *Application Layer* specify the messages to send over the *Uhura Network* and accept the incoming ones. It is important to highlight that not all of the layers are mandatory: Sometimes, if only one adapter is running, the *Network Layer* can communicate directly with the *Application Layer* without the other two layers. Overall, the resource-consuming, in terms of network performances, and optional is the *Service Layer* since it leverages the abstraction of the network to a level that sometimes is too overstructured for robotic scenarios.

The logic architecture of *Uhura* framework is depicted in Figure 6.2. Internally, it relies on a publish/subscribe paradigm between each *Uhura Adapter* and a central module called *Uhura Core*. The latter is the entry point for the upper layer applications (called generic application modules in the Figure) that need to receive/send data to other nodes of the swarm or the integrated IoT robotic system. A generic application module can request for a new message to be sent by publishing its message on the topic `send_bits_string_data` (binary data) or the topic `send_string_data` (string data). Similarly, it can receive messages by subscribing to topics `received_bits_string_data` or `received_string_data`. Under the hood, the core is in charge of selecting the available adapters to transmit the message on every single hop of the path toward the destination; this is implemented by employing another dedicated topic that each adapter has previously subscribed for outgoing and incoming technology-specific data. In the following, we detail the operations of the *Uhura Core* and *Uhura Adapters*.

6.1.1 Uhura Core

The *Uhura Core* is in charge of handling all the communication requests from a generic, upper-layer application and selecting -in a dynamic way- the most suitable adapter to use for the data transfer. Similarly, on the reverse channel, the module receives data from the adapters and forwards a parsed version of them to the generic application on top. To this aim, the *Uhura Core* has been designed with a modular architecture as depicted in Figure 6.3.

Its five modules are:

- **Message Handler**: The *Uhura Core* introduces a structured message format including the payload, type, timestamps, destination, and other metadata useful for the Router

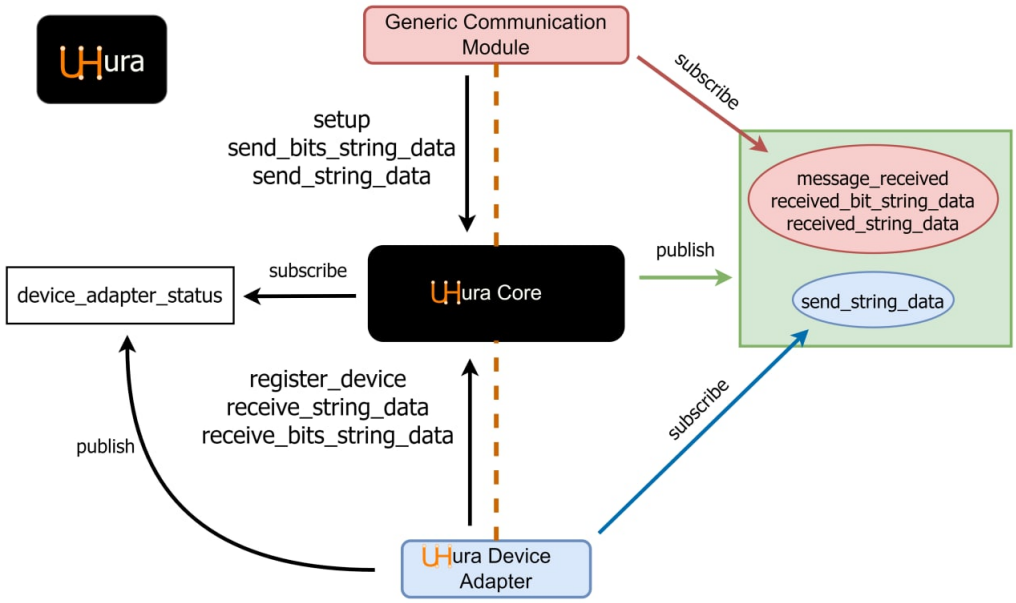


Fig. 6.2 *Uhura* publish/subscribe architecture

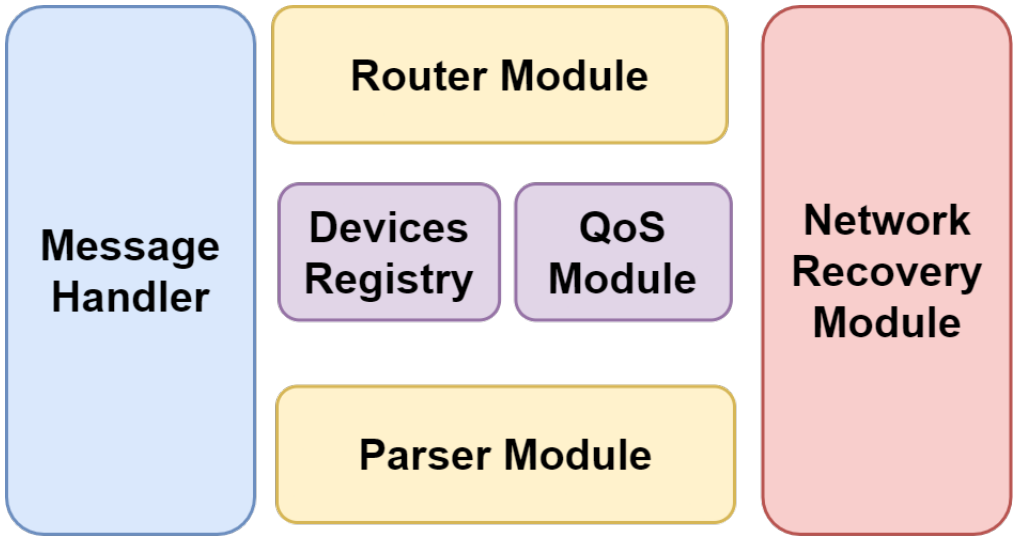


Fig. 6.3 *Uhura Core* architecture

Module. The metadata is added to each outgoing message received from the generic application module; each message generated in this way is then transmitted to the Router Module.

- **Router Module**: The module is in charge of determining the end-to-end path towards the *Uhura* destination. To this purpose, we assume that each *Uhura* node is assigned to a unique identifier. The module receives the list of active neighbors from each

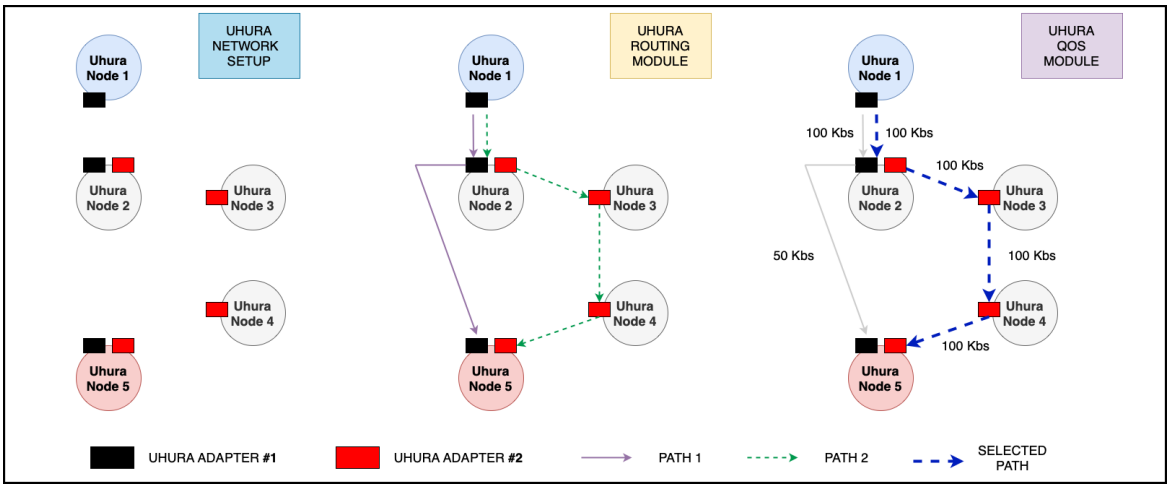


Fig. 6.4 The operations of the *Uhura Core* components on a simplified network robotic scenario with 5 Nodes. Node 1 is the source; Node 5 is the destination; all nodes are provided with a single adapter except for Node 2 and Node 5. The Routing Module (Figure in the center) establishes two different paths. The QoS Module (Figure on the right) takes into account the average throughput on each link and selects Path 2 since it maximizes the performance on the bottleneck link.

Uhura Adapter. It runs a routing algorithm to discover single or multiple paths toward the destination, as depicted in Figure 6.4. Note that a path can be composed of links mapped to different *Uhura Adapters*. In addition, the module collects QoS statistics for each link provided by the Channel Manager described later in Section 6.1.2.

- **Device Registry**: Each adapter during its startup must register itself to the *Uhura Core* through the Devices Registry. The module stores a list of available devices and their tech-specific addresses. In addition, it keeps the current status of the adapters by collecting periodic feedback about their operations.

- **QoS Module**: The QoS module receives the list of paths towards the *Uhura* destination, which the Router Module has computed. Then, it selects the most suitable path according to the QoS requirement of the generic application. Multiple policies can be used at this stage. For the case of Figure 6.4, the path with maximum throughput on the bottleneck link is selected. In Section 6.4 we evaluate the Qos Module's performance for a multi-stack single-hop scenario where the policy is to maximize the network reliability.

- **Parser Module**: The Parser is in charge of transforming generic data from an adapter or a generic application into a valid *Uhura* message using an interface definition

language(IDL)[88] like `Protocol Buffers`[1] or `OMG IDL`[2]. Data generated in this way can be quickly pushed and read if there is an intra-process system like `ROS` [89] or `NATS`[90] able to process structured messages.

- **Network Recovery Module** enable an *Autonomic Faulty Node Replacement* system for WSNs deploying UAVs to heal the network or increase the overall coverage and reliability of the network. This module has a dedicated section 6.5 since was deeply investigated and validated.

6.1.2 Uhura Adapter

The adapter wraps the device API of a specific antenna module (e.g., XBEE 900 pro or any transmission module) and can receive an adapter-specific request from the core or directly from an external source. Eventually, a generic application can invoke the adapter directly, e.g., to minimize the processing latency or to reduce the software layers on low-power devices, but clearly lose all the advantages provided by the core component.

Figure 6.5 shows the component of each adapter, i.e.:

- **Device API**. It is the specific API of the antenna module attached to the host. This module permits sending and receiving data in plain text or binary format and retrieves all the information about the incoming packets (and all their metadata).

- **SerDes (Serializer/Deserializer)**. As the name suggests, it performs data conversion in order to optimize the payload. In addition, it adds a *packets id* field to the binary payload with a fixed length in order to create a unique combination of *id-host-timestamp* for each packet received The deserializer identifies the packet type and parses the payload that the upper modules will use.

- **Log Manager** All packets transmitted/received are logged into a file created during the adapter startup. The log has the same structure for all the *Uhura Adapters* so that it can be easily processed and analyzed by external scripts, again regardless of the specific M2M stack in use.

- **Channel Manager**. The module allows the computation of the QoS wireless link. To this purpose, it periodically transmits heartbeat messages with its own Uhura-id to all other available devices in a 1-hop neighborhood. The output is used by the *Uhura Core* and specifically by the Routing and the QoS modules. we remark that the link

[1] https://developers.google.com/protocol-buffers
[2] https://www.omg.org/spec/IDL

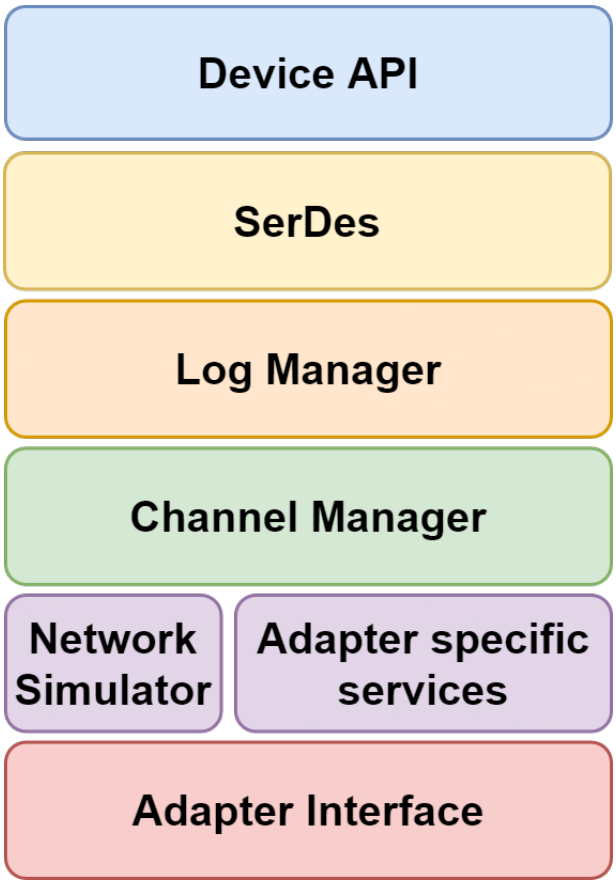


Fig. 6.5 *Uhura Adapter* layered architecture

performance estimation can be done with real traffic transmitted on that link as well and that the heartbeat mechanism is used only in the absence of it.

- **Network Simulator**. This optional module is used if a new technology is available, but the performance limits are unknown. To this aim, the module provides a collection of basic configurable tests to simulate the channel traffic.

- **Adapter Interface** This interface implements all services and topics presented in the software architecture overview. A *Uhura Adapter* must expose the topics and services needed by the *Uhura Core* to send and receive parsed data. Moreover, it should provide access to the Channel Manager for gathering information about the link performance. As for the Log Manager, the adapter's interface is uniform across all the M2M stack supported.

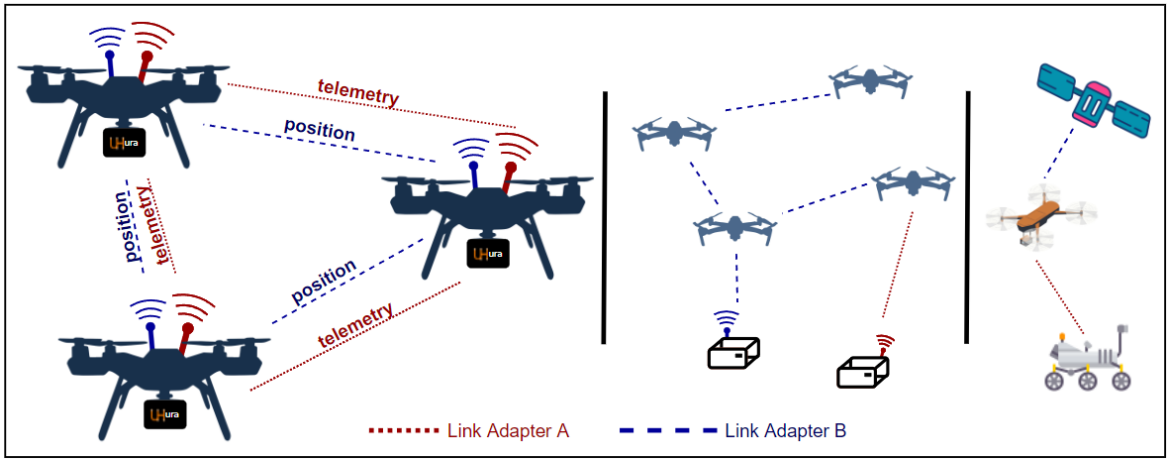


Fig. 6.6 Three use cases of the *Uhura* framework: the left side shows a small-scale UAV swarm exchanging data by using two different adapters module based on type of the message. The subfigure in the center shows an aerial swarm connecting two IoT gateways on the ground by exploiting two different adapters on the air-to-ground link. The subfigure on the right shows a UAV acting as a network bridge between a UGV and a satellite.

6.2 Use cases

The *Uhura* framework can be used on a wide set of scenarios. The most straightforward one is when a single M2M technology is available on each device of the swarm. In this case, the *Uhura* node can communicate by using the adapter of that technology without employing the core. At the same time, the combined usage of multiple M2M technologies within the same scenario can give more flexibility to the swarm deployment or may represent the only solution to communicate with specific IoT devices, as mentioned in the Project Chapter. we remark that *Uhura* supports multi-stack scenarios natively by hiding all the networking details to the application layer, for which only the virtual *Uhura* network is visible. In the following, we discuss three possible use-cases for the *Uhura* framework, also depicted in Figure 6.6.

6.2.1 Multi M2M stacks swarm

In this use-case, a subset or all the nodes of the robotic swarm are equipped with multiple M2M stacks. We assume that M2M stacks are used concurrently but for different applications by taking into account both their QoS requirements and the characteristics of each wireless technology. In the example of Figure 6.6, the swarm uses two *Uhura Adapters*. The Blue adapter is used to broadcast position messages, while the Red adapter is used to exchange telemetry messages about the UAV. we consider position data critical for swarm maintenance;

for these reasons, they should be sent as soon as possible with a stable connection. The *Uhura Core* would select the best adapter available in terms of packet delivery ratio. Vice versa, the telemetry application uses the less reliable channel since packet losses are not as critical as in the previous case.

6.2.2 Swarm connecting multiple base stations

In this scenario, the swarm must collect data from multiple IoT gateways that are mapped to different M2M technologies. Differently from the previous case, only a few UAVs need to support multiple adapters (only one in the Figure), while the rest of the swarm can use a single adapter for intra-mesh communications.

6.2.3 Network bridging

Similarly to the previous case, a node of the swarm acts as a bridge across multiple M2M stacks and enables communication from other nodes of the swarm. Specifically, in Figure 6.6 the swarm is composed of one UAV and one UGV. The UAV supports two *Uhura Adapters*, one for the UGV link and the other for the satellite link. Thanks to the Routing Module of the *Uhura Core*, the UGV can exchange data on the satellite link via the UAV node.

6.3 Implementation

As mentioned in the previous Section, the *Uhura* architecture is based on a publish/subscribe environment provided by the host. In our case, we implemented and tested a preliminary version of the Uhura framework using ROS Noetic[3] as an intra-process communication enabler: although designed for robotic scenarios, it can be used as a standalone framework on every application. Since the `ROS` community moved to its second version, `ROS2`, we decided to use `NATS` as intra-process communication in combination with `Protocol Buffer` as IDL for the current Alpha Version. This decision was made to expand the possibility of writing new adapters taking into account the numerous implementation of the NATS client.

6.3.1 Uhura Core

The *Uhura Core* is a NodeJs [91] application. The current implementation includes a subset of the modules presented in Section 6.1. More specifically, it supports the registration of a *Uhura Adapter* and the message input/output to/from all the network transceivers available

[3] http://wiki.ros.org/noetic

on the host. The Router Module supports only single-hop communication while the QoS Module implements a policy so that it switches to a different *Uhura Adapter* in case the Channel Manager reports errors or a high number of packet loss. In addition, it supports multiple active network connections at the same time.

6.3.2 Uhura Adapter

Each adapter is a NodeJs application written with the specific API device language. At present, we implemented three adapters, one for XBee 900 pro from Digimesh, one for BLE Mesh[92] using a nordic nRF52840 DK, and one for NATS over TCP/IP.

Uhura Adapter XBee

The adapter for XBee has been implemented in Python3 thanks to the Device API provided by Digimesh. It supports dedicated "udev" rules for its devices to bind the port. For example, a device XBee can be found in the list as *uhura_XBEE_DEVICE_n*. The adapter works with the antenna's standard version and the programmable version.

Uhura Adapter bleshjs

The adapter communicates with custom firmware using BLE Mesh SDK flashed on an nrf52840 called *Blesh*. If provisioned in a proper way, it can exchange plain text messages thanks to a vendor model implemented on it. The firmware can be interfaced with the Nodejs package called `bleshjs`[4]; so the adapter is a NodeJS process.

Uhura Adapter NATS

This adapter allows the exchange of messages using a NATS cluster. Basically, two or more Uhura Nodes in the same TCP/IP network, like WiFi, are servers of the same NATS cluster. This is an improved Socket TCP/IP communication version since NATS handles all the high-level networking aspects. In this case, there is no serialize/deserialize phase because the core itself uses the same system and data structure. In case clustering is not possible, the Socket basic version is used. Real-world experiments were performed using a multi-rotor system developed by the company Fly4Future and running the open source software stack release by the Multi-robot Systems Group at the Czech Technical University in Prague [93] and called MRS System (MRS450). Regarding the hardware, we used the UAV of Figure 6.7 consisting of a DJI f450 frame with four rotors, a Pixhawk4 autopilot, and an Intel NUC

[4]https://www.npmjs.com/package/@patonz/bleshjs

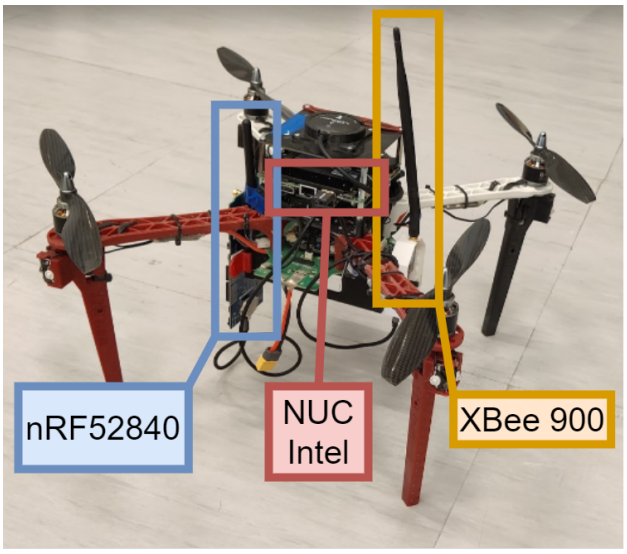


Fig. 6.7 UAV equipped with XBee 900 pro, nRF52840 and a NUC Intel

companion board for controlling the mission flow and running the *Uhura* framework. In addition, we employed sensors such as a rangefinder for height above ground estimation, a GPS module for self-localization in the test arena, and two wireless radio transceivers mounted on the UAV: a Nordic nRF52840 board with `BLESH` firmware and a DigiXbee 900 pro for RF 900Mhz communication.

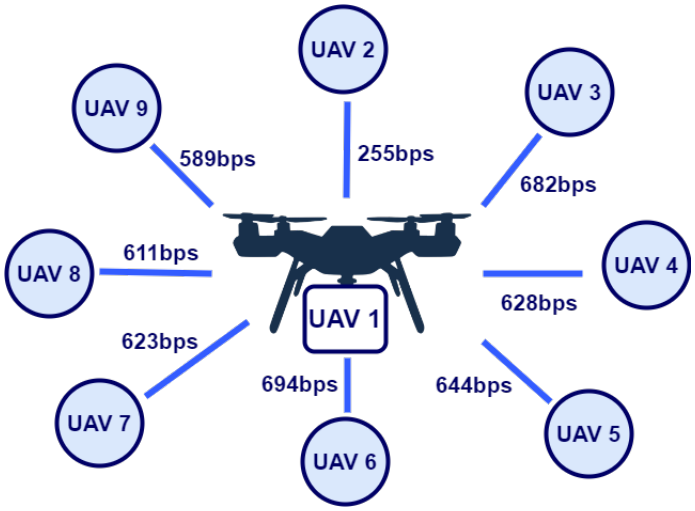


Fig. 6.8 A schematic view of the quality of all the wireless links available on $\texttt{UAV}_1$.

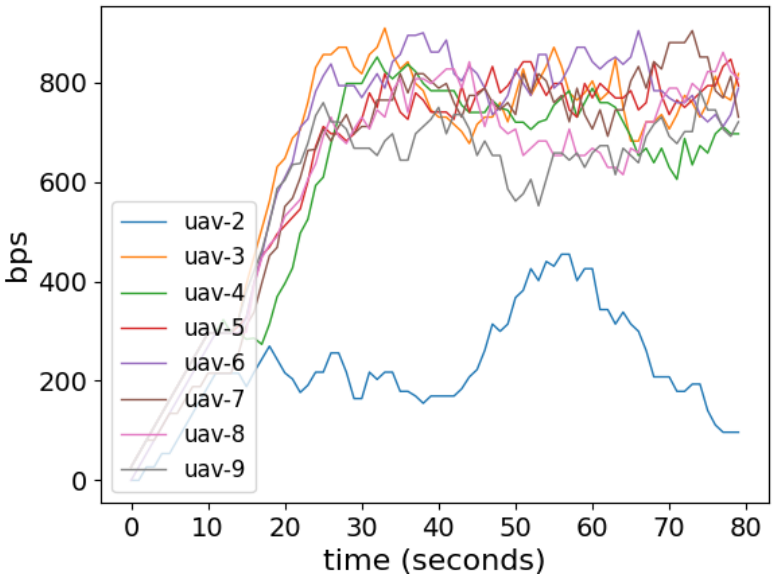


Fig. 6.9 The QoS of the links over time for $\texttt{UAV}_1$

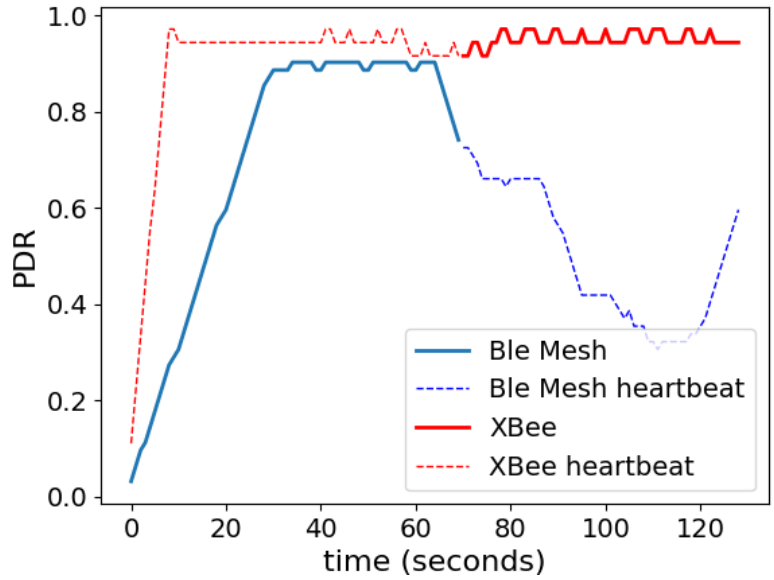


Fig. 6.10 Double Connection experiment: during the first part of the experiment, the Bluetooth connection is active and used, while the sub-GHz link is active during the second part.

6.4 Validation and Experimental Results

we validated the operations of the *Uhura* framework through two studies: i) a test bed using a UAV swarm where each drone is equipped with only one wireless interface/adapter, and ii) a test bed using a single drone and a ground gateway where both the devices are equipped with two different wireless interfaces/adapters. The studies allow me to evaluate different functionalities and components of the *Uhura* framework. The first test described more in detail in Section 6.4.1, shows the framework's capability, specifically of the Channel Manager module, to collect and analyze the performance metrics related to the network quality of the available adapters. The second test, detailed in Section 6.4.2, investigates the operations of the QoS Module described in Section 6.1 and its ability to select the best adapter based on a desired policy.
The experiments have been performed by using the aerial hardware/software platform described in Section 6.3.

6.4.1 Uhura Channel Manager

In this test, we considered a swarm composed of 9 UAVs, $\{\texttt{UAV}_1, \texttt{UAV}_2, \ldots, \texttt{UAV}_9\}$, whose task is to fly while keeping a predefined static formation. The rationale of the formation algorithm is not the focus of this test; rather, we investigate the ability of each drone to monitor the available connections in order to take relative actions based on the QoS of the links. Regarding the communication workload, the formation algorithm relies on a continuous exchange of 1-hop broadcast messages of `HELLO` packets that contain, among other fields, the position information used for the UAV's movements. The message's size is 20bytes; a new message is broadcasted every 200ms. As described in Section 6.1, the Channel Manager module plays an important role in monitoring the QoS on each available *Uhura Adapter*. In this experiment, we focused on the throughput index to monitor the quality of the connections between each pair of neighbor UAVs.
Figure 6.8 depicts an example of connection monitoring for $\texttt{UAV}_1$ while flying in formation. Shows that the UAV has eight links representing the wireless connections with the other UAVs belonging to the swarm. For each link, we report a snapshot of the actual QoS, i.e., the throughput expressed in *bps*. Here, $\texttt{UAV}_1$ has good connections with all the UAVs except for the $\texttt{UAV}_2$. For the same flying experiment, Figure 6.9 shows the QoS of the eight links over time. Due to flight maneuvers inside the swarm, it is possible to notice a high throughput variation for the link $\texttt{UAV}_1 \leftrightarrow \texttt{UAV}_2$. Here is possible to see how the application algorithm has all the information needed to monitor all the connections. After 60sec of flying, we notice

that the formation algorithm was able to improve the total amount of throughput for $\texttt{UAV}_1$, while, on the contrary, the connection $\texttt{UAV}_1 \leftrightarrow \texttt{UAV}_2$ is getting worse.

6.4.2 Uhura QoS Module

In this second experiment, we analyze the QoS module's operations capable of orchestrating the presence of multiple *Uhura Adapters*. The testbed is composed of one UAV and one ground gateway (GW). Both devices are equipped with two network interfaces: the sub-GHz and the Bluetooth Mesh. After the take-off, the experiment works as follows: the UAV moves toward the GW. As soon as the Bluetooth Mesh connection is activated, the GW starts sending the sensor data to the UAV. If the connection becomes unstable, the transmission switches to the sub-GHz technology that works as a backup link. The rationale behind this test is the emulation of an application scenario where multiple ground sensors, and one UAV, are connected through the Bluetooth Mesh network technology. In such an environment, the UAV serves as a data mule, i.e., it flies over all the ground sensors in order to collect all the data through the Bluetooth Mesh links. Here, a subset of ground sensors acts as gateways to send data to the UAV. Due to the high speed of the UAV and the short-range connectivity of the Bluetooth links, we envision the possibility of adding long-range communication - like the sub-GHz technology - on both the UAV and all (or a subset of) GWs. As a result, the long-range communication will serve as a backup channel in case the data transmissions on the Bluetooth link become unstable or unavailable due to the UAV movements.
The key role in this experiment is the QoS Module of the *Uhura* framework that decides which adapter to use if multiple connections are available. The adopted policy for this experiment is described by the pseudo-code in Algorithm 5.

Algorithm 5: Policy adopted by the QoS Module

input : i_1, i_2, thr
output : out
1 $out \leftarrow i_1$;
2 **if** $QoS(i_1) < thr$ *and* $QoS(i_2) \geq thr$ **then**
3 $\quad out \leftarrow i_2$;
4 **end**
5 **return** out;

Here, i_1 is the Bluetooth Mesh interface, and i_2 is the sub-GHz one. QoS is the function that calculates the actual quality index of the specific interface. In this test, we used the

Packet Delivery Ratio (PDR) as the channel index to evaluate the channel reliability. we put as an example a quality threshold ($thr = 0.8$) in the experiment to decide when to switch to the backup interface.
The experiment results are depicted in Figure 6.10. Here, the blue line indicates the Bluetooth connection, while the red indicates the sub-GHz link. Moreover, the dashed lines indicate the control packets' PDR, whereas the continuous line indicates the data packet's PDR. As explained in Section 6.1, the *Uhura* framework provides a heartbeat control message to track each adapter's QoS continuously. The Channel Manager uses these control messages to monitor the available interfaces when no data is transmitted on them.

It is possible to notice that, on the left part of the Figure, the PDR of Bluetooth increases as a consequence of the connection establishment process. When the UAV is close to the GW, the PDR is above the *thr*, and hence the transmission continues using the Bluetooth Mesh adapter. As soon as the UAV moves out from the Bluetooth range, the PDR drops below the threshold value (see around the second 70). At this point, the QoS Module triggers the policy action and activates the backup connection over the sub-GHz channel. We can see that the quality of the Bluetooth link does not improve anymore; hence the wireless transmission completes at a second 130 by using the backup connection.

The dashed lines in Figure 6.10 show the PDR of the heartbeat message on both the technologies, blue for Bluetooth and red for sub-GHz. These lines reveal the ability of the framework to keep monitoring the idle interfaces to be prepared in case of need.

6.5 Autonomic Faulty Node Replacement

In this work, we delve into the application of Unmanned Aerial Vehicles (UAVs) in the field of Wireless Sensor Networks (WSNs) for resilient communications. With the increasing need for reliable data transmission in remote and inaccessible areas, UAVs can be used to bridge the gap between ground-based sensor nodes, thereby increasing the overall coverage and reliability of WSNs.

One of the major challenges in WSNs is the replacement of faulty nodes, which can occur due to various factors such as battery exhaustion or physical damage. To address this issue, we propose an autonomic replacement system for faulty nodes in UAV-assisted WSNs. The proposed system utilizes a test-bed to simulate the replacement of faulty nodes in real-time, and evaluate its performance.

To support the proposed system, we draw upon various studies and literature on the Internet of Things, UAVs, and WSNs. For example, [94] presents an overview of the enabling

technologies, protocols, and applications of the Internet of Things, while [95] explores the use of UAVs for air quality sensing in smart cities. Additionally, [96] examines the data collection mechanism of WSNs based on UAVs, and [97] investigates the use of UAVs for energy consumption minimization in WSNs.

Furthermore, [98] provides a survey of medium access control protocols for UAV-aided WSNs, [99] presents a case study on the use of UAVs for ultra-low-power monitoring systems, [100] discusses the joint optimization of UAV trajectory, altitude, velocity, and link scheduling for minimum mission time in UAV-aided data collection. Moreover, [101] investigates the UAVs assisted network partition detection and connectivity restoration in wireless sensor and actor networks, [72] explores the self-organizing aerial mesh networks for emergency communication, [102] presents a placement learning for multi-UAV relaying using Gibbs sampling approach and [103] studies the optimization of energy-latency trade-off in sensor networks with controlled mobility.

Overall, this work presents a novel approach to addressing the challenges of faulty node replacement in UAV-assisted WSNs and provides a comprehensive analysis of related literature and studies.

6.5.1 Network Recovery Algorithm

The main envisioned scenario for UAV-assisted WSNs is where the sensors data is collected by an elected cluster head (statically or dynamically selected among the network nodes) and then offloaded to one or more UAVs physically transporting them to a remote server not directly connected to the WSN [96]. In this work, we assume that the *Uhura* framework is installed on every device and that a sink node - statically selected among the ground devices - is continuously collecting the sensory data from the WSN. Periodically, a UAV fleet, composed of one or more UAVs, visits the WSN to gather the data from the sink node. During the WSN lifetime, some sensors may run out of energy or just disconnect from the other peers due to hardware failures. In such cases, multiple regions of the WSN could be disconnected due to the multi-hop topology. Our proposal envisages that the sink node is in charge of detecting the occurrence of a network partition and activating a recovery procedure; at the end of it, faulty sensors are replaced with UAVs working as mobile routers. In the following, we detail the recovery procedure running on the Recovery Module within the *Uhura* Core (Figure 6.11).

Let $G_0 = \{S_0, E_0\}$ be the overlay graph at deployment time, where $S_0 = \{s_{\text{sink}}, s_1, s_2, \ldots, s_N\}$ is the set of $N+1$ deployed sensors, with s_{sink} defined as the sink node, and $E_0 = \{w_0^{i,j} | s_i, s_j \in S_0\}$ is the set of weighted edges connecting the sensors. The edge weight $w_0^{i,j}$, with $0 < w_0^{i,j} \leq 1$, defines the quality of the network connection based on the Packet Delivery

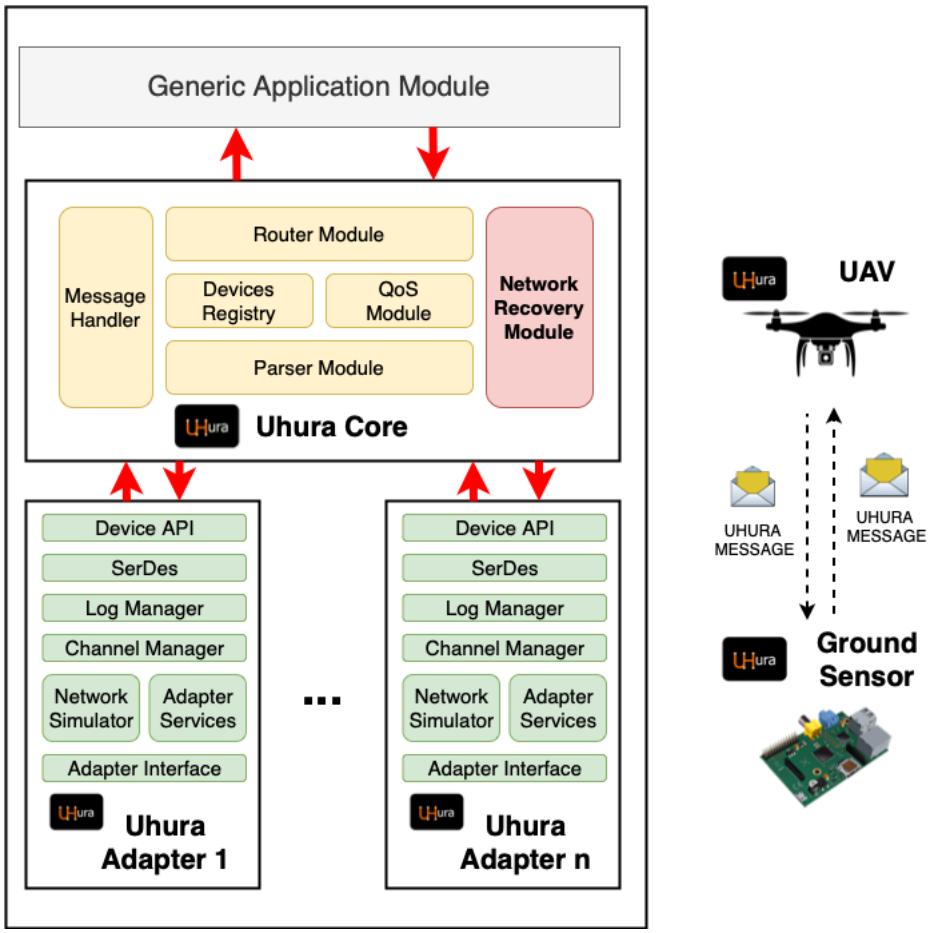


Fig. 6.11 The architecture of the *Uhura* framework with the Recovery Module.

Ratio (PDR). Such metric is periodically computed by the QoS Module of the *Uhura* Core. We assume that the graph G_0 is calculated, and then frozen, after the initial deployment of the WSN.

We assume that at time t, a UAV fleet $U = \{u_1, u_2, \ldots, u_M\}$ of M UAVs, establishes a connection with the sink node s_1 to download the sensory data. After completing the data transfer, the sink node activates the recovery procedure from the Recovery Module in the *Uhura* Core. The procedure includes the following steps (see Figure 6.12):

1. Calculate $G_t = \{S_t, E_t\}$ as the actual overlay graph at time t, with $S_t \subseteq S_0$. Let $G_F = \{S_F, E_F\}$ be the inactive graph where S_F is the set of disconnected sensors, with $S_F \cup S_t = S_0$, $S_F \cap S_t = \emptyset$, and $E_F \subset E_0$ are the arcs connecting only the inactive nodes. If $S_F = \emptyset$ then ends the recovery procedure.

2. Calculate $S_C \subseteq S_t$ as the set of candidate inactive sensors to be replaced by a UAV: $s_j \in S_C$ only if $\exists s_i \in S_t$ such that $w_0^{i,j} \in E_0$, i.e., if the active sensor s_i at time t were

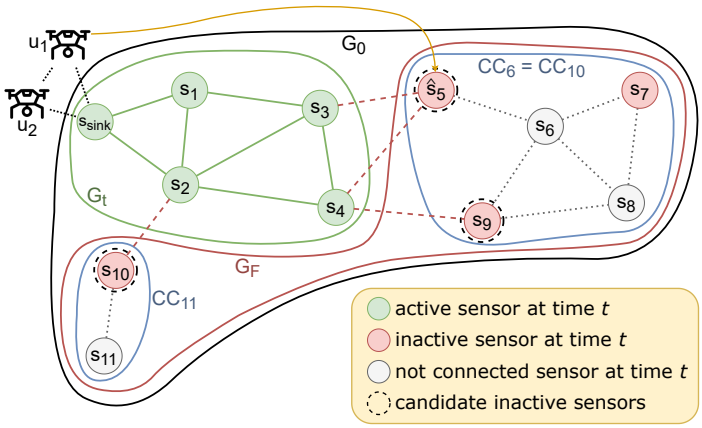


Fig. 6.12 WSN scenario and the setup of the proposed network recovery strategy.

directly connected to the inactive sensors s_j at deploy time. Here, let $w^j_{0\leftrightarrow t}$ be the maximum edge weight from E_0 connecting the inactive node $s_j \in S_F$ with any sensor in S_t.

3. For each $s_j \in S_C$ calculate in G_F the connected components set CC_j to which s_j belongs to. We use a Breadth-First Search (BFS) graph visit starting from s_j to determine the connected components.

4. Select $\hat{s}_j \in S_C$ as the chosen sensor to be replaced by the UAV, where $\hat{s}_j = \arg\max_{s_j \in S_C} |\mathrm{CC}_j| + w^j_{0\leftrightarrow t}$. In case multiple sensors have the same connected components set size, we choose the sensor with the maximum value of $w^j_{0\leftrightarrow t}$.

5. Choose randomly one available UAV from the UAV fleet and send it to the $\hat{s}_j$ coordinates in order to replace the operations of the inactive sensor. The target UAV will land at that position and re-broadcast each received *Uhura* message on each adapter.

6. If at least one UAV is left, then iterate the procedure starting from point 1.

6.5.2 Performance Evaluation

we evaluated the proposed network recovery strategy in a small scale test-bed deployed at the drone arena of the TII [5] (Technology Innovation Institute, Abu Dhabi). The test-bed included one ground WSN and one drone. The ground network consisted of four Raspberry PI 4 devices, each provided with a Nordic nRF52840 BLE interface with the Bluetooth mesh stack. we created a multi-hop Bluetooth mesh WSN as depicted in Figure 6.15: nodes $S1$, $S2$

[5] https://www.tii.ae/

and $S3$ transmitted periodic measurements towards the sink, with nodes $S1$ and $S2$ working as mesh relay nodes. The drone was a DJI f450 frame with 4 rotors, a Pixhawk4 autopilot, and an Intel NUC companion board. The sink node was connected via cable to a laptop device which we used to collect the network logs and compute the performance metrics. Both the drone and all the ground devices ran a local instance of *Uhura* with a BLE adapter[6].

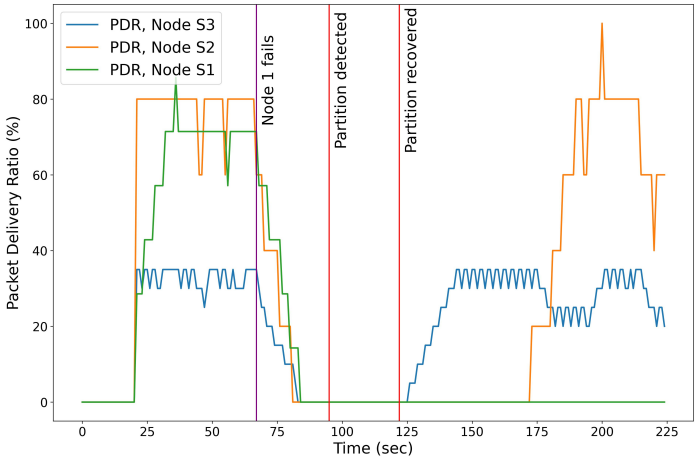


Fig. 6.13 PDR for each data flow between nodes S1, S2 and S3 and the ground sink before and after the recovery procedure.

Balena Cloud[7] was used to ease software deployment. In my experiments, we forced the failure of node $S1$, which at time t=70 seconds disconnected from the WSN and stopped sending its own and relayed messages towards the sink. As a result, the recovery procedure was started, with the sink detecting the network partition at time t=95 seconds and issuing a command to the UAV; the latter landed close to node $S2$ at time t=125 seconds (see Figure 6.13). The landing coordinates were hardcoded in the control software of the drone, leaving the optimization of the mobile sink placement as future work. Figure 6.13 shows the PDR computed at the sink node for the three different data flows originated from nodes $S1$, $S2$ and $S3$. The PDR was affected by the source node position: for this reason, $S3$ experienced the lowest PDR before node $S1$ failure due to the two hop distance from the sink. After node $S1$ failure (t=70 seconds), the data collection was completely disrupted for around 50 seconds; after $t > 125$ seconds, it was easy to notice that the data flows of nodes $S2$ and $S3$ were fully re-established and the PDR metric achieved similar results than before the node failure. However, the message relay operations were now operated by the UAV landed on the ground rather than by node $S1$. Figure 6.14 shows the Received Signal Strenght (RSS) of the wireless link between nodes $S1$, $S2$, $S3$ and the UAV. It was easy to notice that the

[6]https://github.com/patonz/uhura/tree/main/src/lib/adapters/blesh
[7]https://www.balena.io/

RSS values before and after the recovery procedure were similar, but with higher fluctuations when the UAV relay was active. This was due to the different UAV antenna as well as the varied channel conditions.

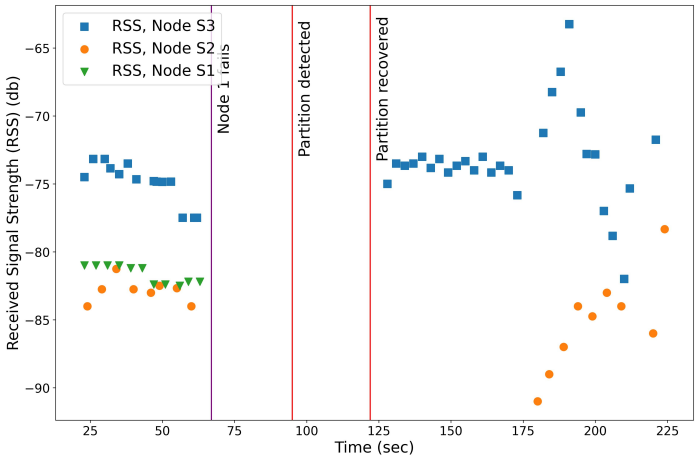


Fig. 6.14 RSS between each node and the UAV.

6.6 Distributed Service Discovery System

As *Scalable* requirement described in Section 3, we want to describe a Distributed Service Discovery System created as *Service Layer* of Uhura. In this layer, novel features were designed and implemented, allowing nodes to discover and execute custom services provided by other peers in a Uhura Network. More specifically, it is a distributed service discovery protocol designed for Robotic Systems-of-Systems, implemented and tested using the Uhura Framework.

6.6.1 Scenario Model

we consider a generic RSoS scenario like the one depicted in Figure 6.15. The nodes composing the robotic swarm can be unmanned ground, aerial or marine vehicles, or a combination of them; indeed, my solution can be implemented on different types of robots and has been tested on the hybrid ground/aerial robotic scenarios as described in Section 6.6.4. we address heterogeneous RSoS in which robots are provided with different M2M communication technologies; to ensure connectivity within the swarm, we assume that some robots are equipped with multi-radio transceivers, as shown in Figure 6.15. The robot-to-robot interactions are handled through the *Uhura* platform, which supports technology-agnostic data exchange among the robots: assume *Uhura* platform to be installed on each robot, as further detailed in

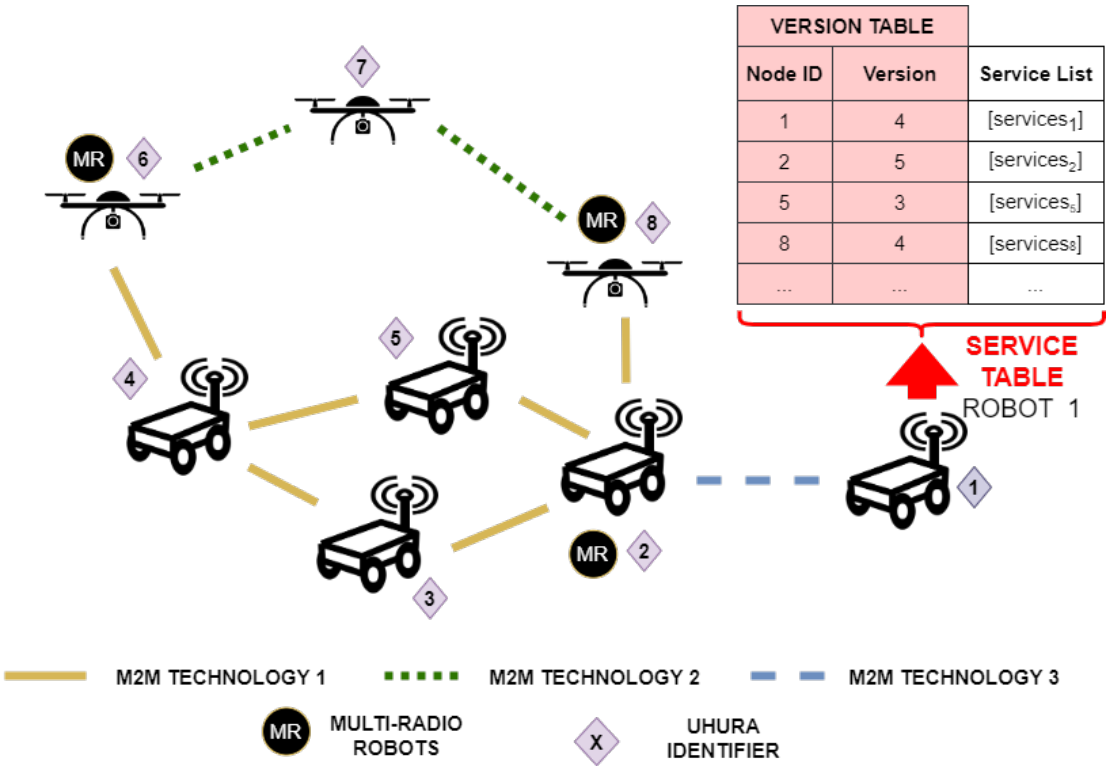


Fig. 6.15 Robotic scenario with heterogeneous *Uhura* nodes.

Section 6.6.4. Some nodes of the swarm may expose network services that can be requested from the other peers; for instance, one robot with higher computational power and energy autonomy may expose a service for ML-based object detection, which may be requested from other robots with constrained computational capabilities. Without loss of generality, we consider in this study two classes of services: (*i*) *state-less functions*, which take some parameters in input and return some values in output (like the aforementioned classification task) or (*ii*) *property values*, which return the current value of a readable variable (e.g., sensor data) available at one robot. we investigate the case where services provided by one robot may change at run-time due to varying conditions during the mission (e.g., a property value change), triggering events, or software updates. The goal of the robotic service discovery is to enable robot 1 of Figure 6.15 to be continuously updated about the services available on robot 2 and other robots of the scenario. To this purpose, we highlight that the existing M2M technologies differ in terms of supported range and throughput [104]; in some cases (e.g., BLE, XBee, LoRa), their network stack imposes strict constraints on the maximum size of the communication frame. For this reason, our solution aims at enabling lightweight service discovery by reducing both the latency and the network overhead in terms of the number of messages and size of the latter.

6.6.2 Proposed Protocol

We consider distributed service discovery, i.e., there is no centralized register, and every robot keeps a *Service Table* (see Figure 6.15) that keeps track of the network services within the swarm. Each entry of the *Service Table* is a triple: $\langle \text{Node ID}, \text{Version}, \text{Service List} \rangle$. Here, the *Node ID* identifies the robot inside the *Uhura* network; the *Version* is the version number of the services offered by the robot with *Node ID*, which represents an incremental counter that keeps track of any change in the service list of the node it refers to; finally, *Service List* contains the list of offered procedures, each with a name, input parameter(s), produced output, and a short description. we assume that each service definition has a maximum size equal to s_{service} byte. Let *Version Table* be the table made by the first two columns only, i.e., $\langle \text{Node ID}, \text{Version} \rangle$, as highlighted in Figure 6.15.

The *Service Table* structure is updated according to Algorithm 6. Due to the lack of a centralized controller, the aim of Algorithm 6 is to keep synchronized the *Service Table* of all the robots of the RSoS via a gossiping technique. Sending the full *Service Table* to the neighborhood in a periodic way would increase the network overhead drastically because: (*i*) there may be unnecessary transmissions when robots are already synchronized; (*ii*) the size of the messages may exceed the size of the maximum frame of the M2M technology in use, hence requiring costly in-network fragmentation operations when supported. For this reason, we introduce a *hashing* function h_{vt} that generates the hash code of the *Version Table*. This latter, in fact, contains the Service Counter, which represents an incremental counter that keeps track of any change in the service list of the node it refers to. For this reason, by comparing the hash code received from a peer with the local hash code, each robot can understand whether a synchronization action is needed.

Algorithm 6: Service Discovery

Input: nodeID, $d_{\max}$

Procedure *ServiceManager*
  vt ← initVersionTable();
  st ← initServiceTable();
  version ← 0;
  h_{vt} ← hash(vt);

Procedure *serviceRegistration_Delete_Update(serv)*
  version ← version +1 vt ← updateVersionTable(vt, nodeID, version) st ← updateServiceTable(st, nodeID, serv, version) h_{vt} ← hash(vt)
  send_SYNC_BETTER(nodeID, $d_{\max}$, [⟨ nodeID, version, $\text{services}_{\text{nodeID}}$ ⟩])

Procedure *tCheckExpires*
  send_SYNC_CHECK(nodeID, $d_{\max}, h_{\text{vt}}$)

Procedure *receiveSYNC_CHECK(rcvID, h'_{vt})*
  if $h'_{\text{vt}} \neq h_{\text{vt}}$ **then**
    send_SYNC_INTEREST(nodeID, rcvID, vt)
  end

Procedure *receiveSYNC_INTEREST(rcvID, rcv_vt)*
  if *rcv_sv* $\neq$ *sv* **then**
    worse_{vt} ← getMyWorse(rcv_vt, vt) l_w ← [ ] ;
    forall $\langle id, v \rangle \in worse_{\text{vt}}$ **do**
      $l_w \leftarrow l_w +$ ⟨ id, v ⟩
    end
    send_SYNC_REQUEST(nodeID, rcvID, l_w) $\text{better}_{\text{vt}}$ ← ;
    getMyBetter(rcv_vt, vt) ;
    l_b ← [ ] ;
    forall $\langle id, v \rangle \in better_{\text{vt}}$ **do**
      $l_b \leftarrow l_b +$ ⟨ id, v, services_{id} ⟩
    end
    send_SYNC_BETTER(nodeID, rcvID, l_b)
  end

Procedure *receiveSYNC_REQUEST(rcvID, l)*
  l_b ← [ ] ;
  forall $\langle id, v \rangle \in l$ **do**
    $l_b \leftarrow l_b +$ ⟨ id, v, services_{id} ⟩
  end
  send_SYNC_BETTER(nodeID, rcvID, l_b)

Procedure *receiveSYNC_BETTER(rcvID, l)*
  forall $\langle id, v, \text{services}_{id} \rangle \in l$ **do**
    if $v > get\_version(vt, id)$ **then**
      vt ← updateVersionTable(vt, id, v) ;
      st ← updateServiceTable(st, id, v, services_{id}) ;
      h_{vt} ← hash(vt)
    end
  end

The algorithm introduces four different control messages to manage the discovery task: $\text{sync}_{\text{check}}$, $\text{sync}_{\text{interest}}$, $\text{sync}_{\text{request}}$, and $\text{sync}_{\text{better}}$. More in detail, we can split the operations of Algorithm 6 into three phases. After the initialization phase (lines 1 - 5), the following steps are executed:

- *difference checking*: every t_{check} seconds, each node in the *Uhura* network broadcasts the $\text{sync}_{\text{check}}$ control message. The latter is re-broadcasted at $d_{\max}$ maximum hops distance; if $d_{\max} = \infty$, then the message will cover the whole network. If a robot is provided with multiple M2M communication technologies and corresponding *Uhura Adapters* (see Section 6.3), the message is broadcasted on all of them. The $\text{sync}_{\text{check}}$ message contains the hash value h_{vt} of the *Version Table* (lines 8 - 9). Without loss of generality, let A and B be two robots of the swarm. When robot B receives the $\text{sync}_{\text{check}}$ message from robot A, it checks whether its hash value h_{vt} differs from the received one. If so, the next phase is started.

- *difference discovery*: in case a difference in the *Version Table* is detected, robot B sends the message $\text{sync}_{\text{interest}}$ to robot A. The message contains the full *Version Table* of robot B (lines 10 - 13).

- *synchronization*: after having received the $\text{sync}_{\text{interest}}$ message from robot B, robot A can check the differences between the local and received *Version Table*. Here, two actions are taken: (*i*) for all services which are outdated (i.e., robot B holds a fresher version), it requests an updated version from robot B by sending the $\text{sync}_{\text{request}}$ message (lines 16 - 16 and 28 - 33); and (*ii*) for all services which are more up-to-date than robot B, it sends their corresponding entries to B using the $\text{sync}_{\text{better}}$ message (lines 21 - 26). The $\text{sync}_{\text{better}}$ message contains the entries of *Service Table* that are more up-to-date; when robot B receives it, it updates the local *Version Table* by synchronizing the service entries (lines 34 - 41).

Moreover, to speed up the synchronization procedure, when a node generates, updates, or disables a local service, it advertises all other peers by broadcasting a $\text{sync}_{\text{better}}$ message (lines 6 - 7). During this procedure, we can notice that the Version is incremented for any kind of service list change: update, delete, or update (line 7).

6.6.3 Architecture & Implementation

One of the main advantages of this protocol is real testbeds realized thanks to Uhura. Thanks to its modularity and scalable architecture, we defined and implemented two new modules for the Service Discovery Protocol: Discovery and Gateway modules.

Architecture

The *Service Discovery* and *Service Gateway* modules manage the services among the nodes belonging to the Uhura network. Differently from other service discovery solutions, a heterogeneous multi-radio robotic network is of paramount importance to minimize both the overhead due to control messages and the time of service discovery. For this reason, we split the service management into two different modules: *Service Discovery*, which is in charge of the discovery and dissemination of the available services, and *Service Gateway*, which eases the use of remote services. The extended architecture is depicted in Figure 6.16

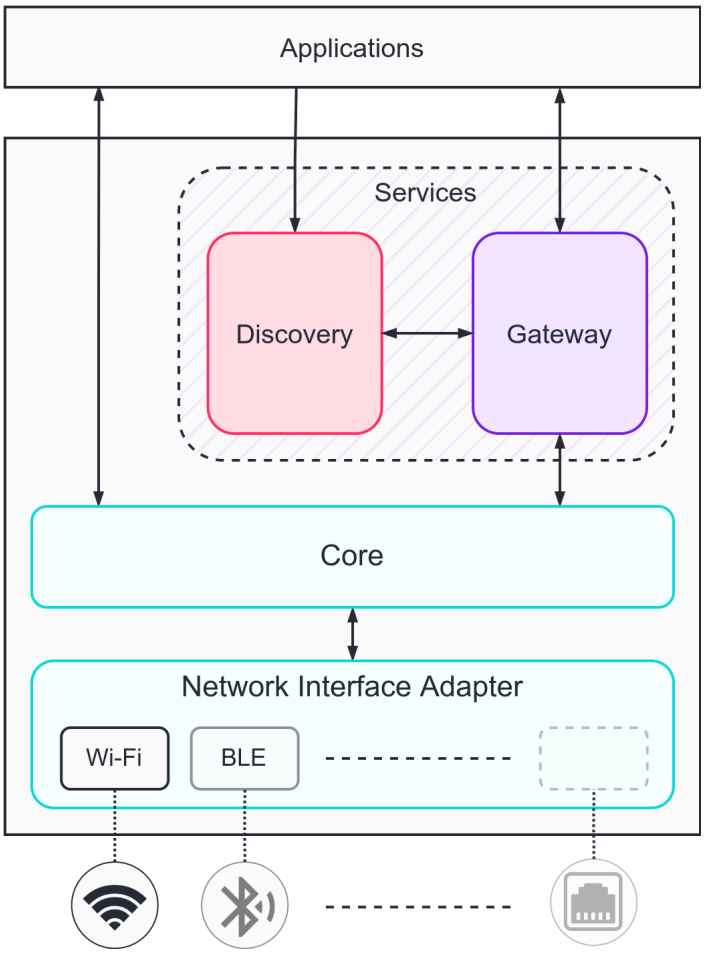


Fig. 6.16 The *Uhura* architecture with the novel modules.

In particular, the *Discovery* module allows both the registration of a new service (or the cancellation and update of a service) and the localization of the services available in the network. More in detail, the *Discovery* module keeps a service table to store all the available nodes in the network and their exposed services. An example is depicted in Table 6.1 that shows the *Service Table* that contains the network services organized in triples:

Service Table		
Version Table		
Node ID	Service Counter	Service List
1	4	[services$_1$]
2	6	[services$_2$]
...	...	...

Table 6.1 The available services discovered. The full table is called *Service Table*, while the first two columns specify the *Service Vector*

⟨Node ID, Service Counter, Service List⟩. Here, the *Node ID* identifies unequivocally the node inside the Uhura network; the *Service Counter* specifies the versioning of the services offered by the node *Node ID*, i.e., every time the node updates the offered services, it increases the *Service Counter* value; finally, *Service List* contains the list of offered services. The latter contains the list of all the exposed services containing its name, the input parameters, the produced output, and its description. We assume that each service definition has size s_{service} byte. Let *Version Table* be made by only the first two columns: ⟨Node ID, Service Counter⟩.

Implementation

The full architecture is implemented in the *Uhura* framework and can be deployed using `Docker`[8]. Every robot of the swarm needs to install at least four local *Uhura* modules, i.e.: the *Core*, at least one *Adapter* for the basic networking functionalities, and the *Discovery* and *Gateway* modules to find and use the custom services offered by other robots. In the testbed described in Section 6.6.4, we used the NATS *Adapter* and the BLE *Adapter*, both in `Nodejs` v16.16[9]. The *Core* is a `Nodejs` application orchestrating the adapters activated on a specific robot.

Discovery and Gateway Both the modules are implemented in `Typescript`[10], and they define Services 6.6.3 and their procedures 6.6.3 according to Protobuff declarations. Also, the API of these modules uses Protobuf as IDL and can be found in the public *Uhura Github* repository [11].

Listing 6.1 Protocol Buffers message declaration of `Service`

[8] https://hub.docker.com/repository/docker/patonz/uhura
[9] https://nodejs.org/en/
[10] https://www.typescriptlang.org/
[11] https://github.com/hyperloris/uhura-discovery/tree/main/src/common/protos

```
message Service {
  string id = 1;
  string name = 2;
  string description = 3;
  string nodeId = 4;
  string modified = 5;
  map<string, Procedure> procedures = 6;
}
```

Listing 6.2 Protocol Buffers message declaration of `Procedure`

```
message Procedure {
  string type = 1;
  string name = 2;
  string description = 3;
  bool reply = 4;
  map<string, string> inputFields = 5;
  map<string, string> outputFields = 6;
}
```

The *Gateway* handles all service requests using a `ProcedureRequest` proto-buffer message 6.6.3. As the name suggests, inside this request, the client must indicate the remote procedure name to call and the sender and receiver *Uhura* IDs. The module leverages the *Core* API to send the encoded version of the proto-buffer to the *Uhura* destination. On the receiver side, the *Gateway* module decodes each request and then invokes the procedure 6.6.3.

Listing 6.3 Protocol Buffers message declaration of `ProcedureReq`

```
message ProcedureReq {
  string receiverUhuraId = 1;
  string senderUhuraId = 2;
  Procedure procedure = 3;
  map<string, string> inputs = 4;
}
```

6.6.4 System Evaluation

This section evaluates the proposed framework using simulations and real test beds. In the first case, we tested the algorithm presented in Section 6.6.2 by simulating a large-scale environment with a variable number of mobile robots in a multi-hop ad-hoc topology. In

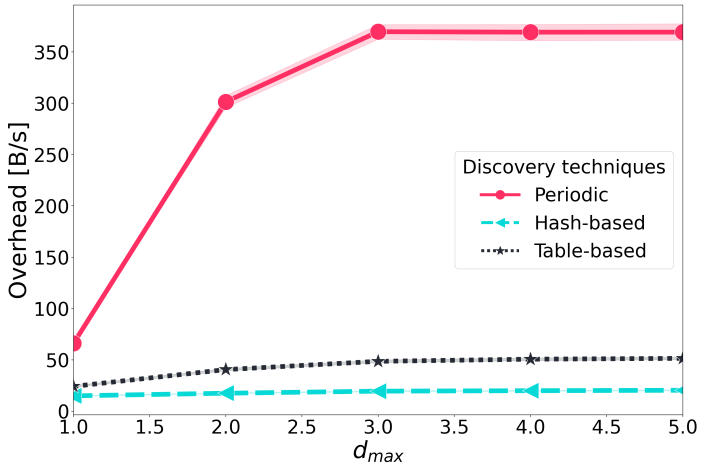


Fig. 6.17 Figure shows the Overhead index by varying the $d_{\max}$ parameter.

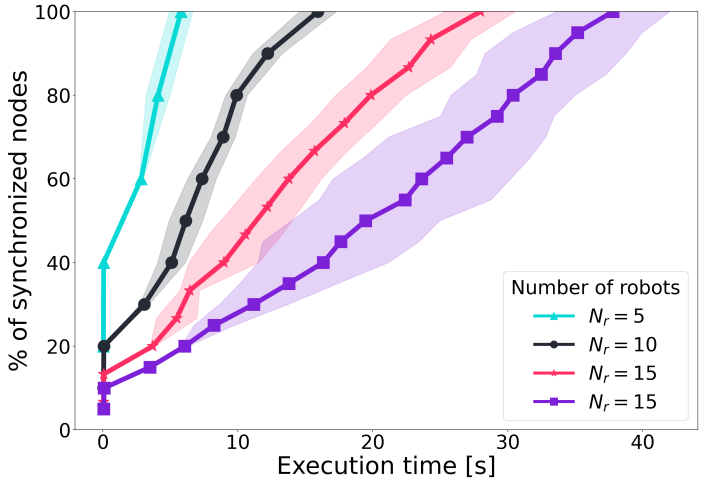


Fig. 6.18 Figure shows the synchronization time of the robots during the execution time.

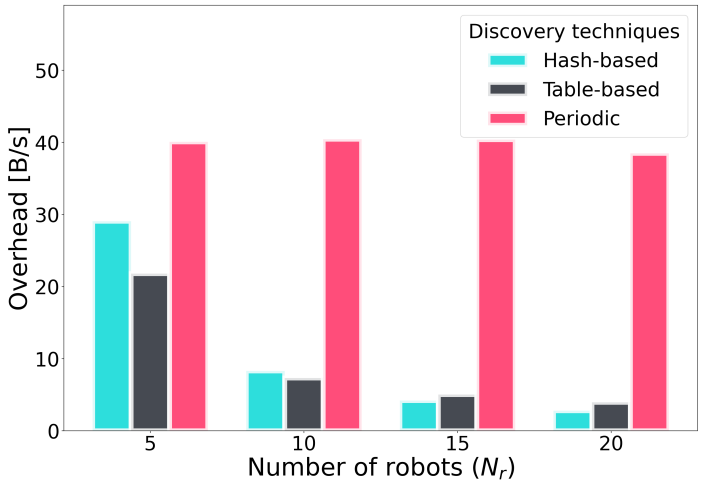


Fig. 6.19 Figure shows the Overhead index by varying N_r in the chain formation test-bed.

Fig. 6.20 Figure shows the robots used for the test beds.

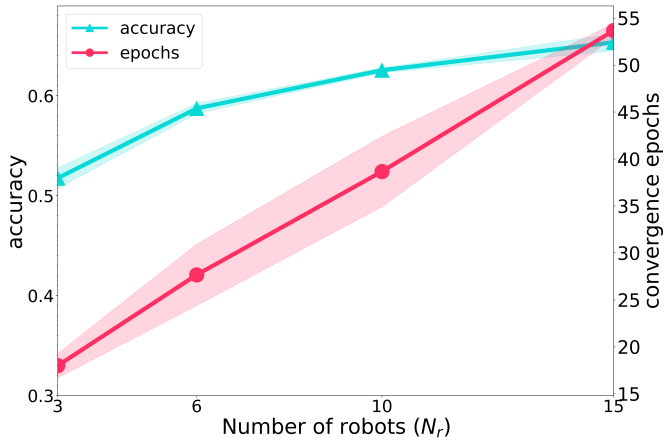


Fig. 6.21 Figure analyzes the FL application's convergence time and accuracy.

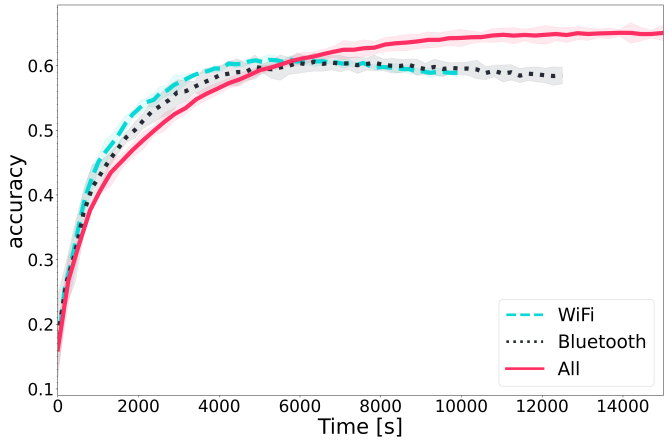


Fig. 6.22 Figure shows the accuracy of the FL application in a multi-radio scenario.

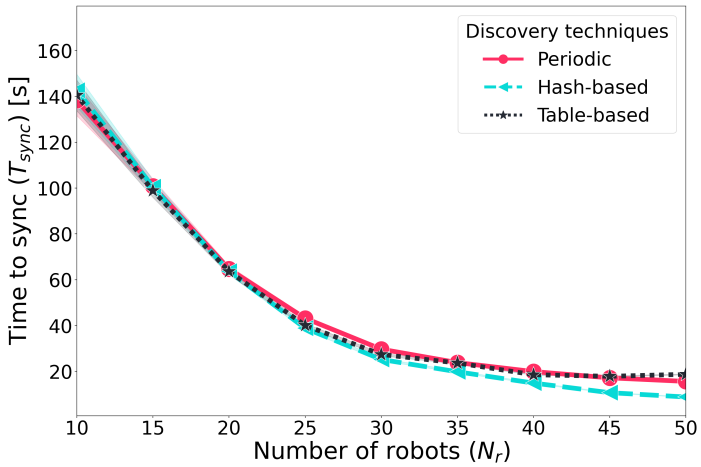


Fig. 6.23 Figure shows the T_{sync} index by varying N_r.

the second case, we considered different robot scenarios aimed to validate the operations of the *Discovery* module and the multi-radio support in a FL use-case. More specifically, we compared three different service discovery algorithms:

- *Hash-based*: this setup corresponds to Algorithm 6 presented in Section 6.6.2.
- *Table-based*: this is our implementation of the algorithm presented in [105]. Different from our proposal, this method does not use any hash function. Hence, it broadcasts the full *Service Table* to discover services' dissimilarities among the robots of the swarm.
- *Periodic*: this method is used as baseline and it consists of an algorithm that broadcasts the full *Service Table* at d_{max} hops every t_{check} seconds.

We compared the algorithms in terms of synchronization time and the network overhead induced by the discovery process. The former is given by T_{sync}, which defines the time elapsed between the starting of the service update procedure and the instant when all the swarm robots have been synchronized. The latter is defined by the variable *Overhead*, which specifies the network load in *B/s* caused by the service discovery task, i.e., the amount of traffic constituted by control messages.

Simulated Large-scale Ad-Hoc Scenarios

Without loss of generality, we model a dynamic search and rescue application where the swarm is used to explore an unknown scenario; depending on environmental conditions, each robot can activate specific pre-loaded sensing or data processing services. To this aim, we

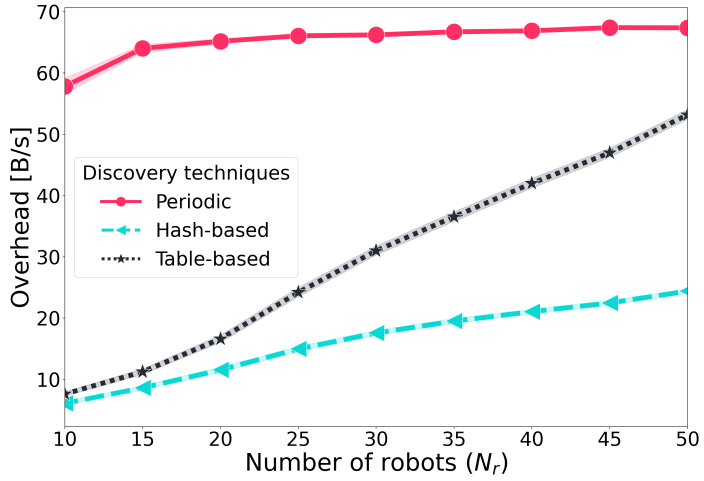


Fig. 6.24 Figure shows the *Overhead* index by varying N_r.

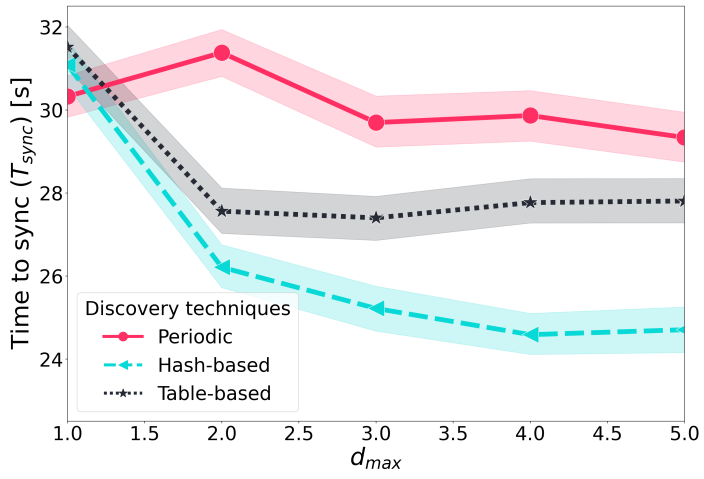


Fig. 6.25 Figure shows the T_{sync} index by varying d_{max}.

assume that the swarm is made of N_r mobile robots exploring a map of 1 $km \times 1$ km. Robots move at a constant speed of v_r meters per second and are able to activate/deactivate a fixed number of heterogeneous N_s network services. To simulate the dynamic loading/unloading of functionalities based on run-time conditions, every $5s$, each robot activates or deactivates one of its N_s services with probability p_s. In this way, it is possible to evaluate the service discovery algorithm in a mobile and time-varying scenario. When not specified differently, the following values are used: $N_r = 25$, $N_s = 5$, $s_{\text{service}} = 128B$, $v_r = 5m/s$, $p_s = 0.2$, $d_{\max} = 1$, $t_{\text{check}} = 5s$. In the first experiment, we analyzed both the synchronization time (T_{sync}) and the network overhead (*Overhead*) when varying the size of the robotic swarm from $N_r = 10$ to $N_r = 50$. The results are depicted in Figures 6.23 and 6.24. We can notice that T_{sync} decreases when increasing the number of robots in the swarm. This behavior is due to the impact of random mobility since temporary network disconnections are quite frequent when $N_r < 30$. For low values of N_r, all the algorithms behave similarly; however, with medium/high robot densities, the *Hash-based* algorithm outperforms the other two algorithms. This can be explained by Figure 6.24 that shows the *Overhead* metric. Indeed, even for high values of N_r, the *Hash-based* algorithm drastically limits the traffic load and hence reduces the risk of network congestion.
In the second experiment, we analyzed the impact of the $d_{\max}$ variable (Figures 6.25 and 6.17) on both the metrics. From the Figures, we can notice that by increasing the number of hops of the dissemination process, the synchronization time is shortened (Figure 6.25) but introducing higher network overheads (Figure 6.17). Similarly to the previous experiment, our algorithm *Hash-based* algorithm outperforms the competitors by limiting the network overhead and reducing the average time needed for the service discovery.

Single-technology Robotic Test-bed

In this Section, we investigate the performance of *Hash-based* service discovery algorithm when implemented in *Uhura* and deployed in small-scale robotic test-beds.
In the first experiment, we deployed a variable number of ground robots placed in a chain formation, i.e., the robots are arranged in a row, and each robot can communicate only with the robot in front of it and the robot behind it. With this configuration, we can evaluate the discovery algorithm in an always-connected multi-hop environment. The ground robots used in our tests are depicted in Figure 6.20. Each node runs an instance of the *Uhura* platform and is equipped with a single Wi-Fi network interface (the multi-radio scenario will be presented later in this Section). Also, each node provides a single network service that needs to be discovered by the other peers. The experiment worked as follows. Initially, we deployed $N_r - 1$ robots; after their synchronization, we introduced a new robot in the

swarm and computed both the synchronization time T_{sync} and the network *Overhead* needed for the robotic swarm to discover the new service belonging to that robot. Figure 6.18 shows the percentage of robots discovering the newly available service over the time of the experiment. When the plotted lines reach the 100% of nodes, then T_{sync} is achieved. we repeated the experiment for different numbers of robots (N_r). At the very beginning of the experiment, when the new robot arrives, the $\text{sync}_{\text{better}}$ message is transmitted immediately to notify the presence of the service; as a result, the first node of the chain become aware of the new service with a fast control messages exchange. Figure 6.19 shows the *Overhead* for discovering the new service. When the chain is short, the *Table-based* method performs better than the *Hash-based* since the size of the $\text{sync}_{\text{interest}}$ message is comparable with the $\text{sync}_{\text{check}}$ message. On the opposite, when $N_r > 10$, our proposal outperforms the other methods.

Multi-technology Robotic Test-bed: Federated Learning

The last experiment considers a hybrid scenario with the ground and aerial robots. The application scenario envisages the presence of $N_r - 1$ ground robots that can gather image data and an unmanned aerial vehicle (UAV) that visits all the ground nodes. Here, we deployed a Federated Learning (FL) service that allows ground devices to train a local model based on their available onboard data and to send the local models (i.e., model's weights) to a server for their aggregation. In this way, there is no need to transmit all the data to a central server and the network traffic is drastically reduced. Also, the privacy of the data sources is maximized. Regarding the FL application service, we propose the distributed training of a model for image classification. For this purpose, we used the CIFAR-10 dataset[12], consisting of 60k 32x32 RGB images belonging to 10 different classes. To emulate the presence of multiple nodes participating to the same federated learning task, we distributed the data over 14 *Uhura*-enabled nodes, and we let only a subset of these nodes actively perform the local training. Figure 6.21 shows both the convergence epochs and the accuracy achieved by the global model after convergence has been reached. In the experiment, each ground robot implements a client FL service while the UAV implements the master FL service. The used devices are shown in Figure 6.20: only the Wi-Fi radio has been used in this experiment. All robots are in the same communication range and thus the UAV can communicate directly with each ground robot. Due to the simple model employed in training, the performance does not exceed 65% of accuracy. However, optimizing the learning accuracy is out of the scope of this work, which focuses on the capability of robots to join and discover the FL task. To this aim, after the discovery of all the ground nodes' services, the master FL

[12]https://www.cs.toronto.edu/ kriz/cifar.html

service, located at the UAV, starts the FL training process. The best performance comes when $N_r = 15$, i.e., all of the 14 ground nodes participate in the federated learning, given the more complete observation that we can obtain at the cost of an increased convergence time. we then performed a second experiment for the following use case: 14 ground robots are placed randomly in a $150m \times 150m$ area. Half of the nodes have a Wi-Fi wireless interface, while the other half can communicate via BLE. The UAV is in charge of periodically visiting these sensors to collect the local models that they have been training through their local observations. The UAV, with a speed of $5m/s$, sequentially visits all the nodes within a travel time of T_{tr} seconds, and it hovers on each sensor to receive its local model within a connection time of T_{conn} seconds. Hence, a single loop to receive all the local models and update the global one lasts for $T_{tr} + N_r \times T_{conn}$ seconds. The mean (μ) and standard deviation (σ) of these time parameters are obtained over multiple preliminary experiments (see Table 6.2).

Table 6.2 Mean and standard deviation of travel and connection times according to the nodes' wireless interface.

Parameter	**Value**
$\mu(T_{tr})$	57.521 s
$\sigma(T_{tr})$	2 s
$\mu(T_{conn})$ (WiFi)	3.697 s
$\sigma(T_{conn})$ (WiFi)	0.949 s
$\mu(T_{conn})$ (Bluetooth)	6.919 s
$\sigma(T_{conn})$ (Bluetooth)	0.02 s

Figure 6.22 shows three different alternatives to the scenario presented earlier: (*i*) only the Wi-Fi nodes participate in the learning, (*ii*) only the BLE nodes participate in the learning, (*iii*) regardless of the wireless interface, all of the nodes participate in the learning. Despite connecting either to Wi-Fi or BLE nodes helps in reducing the convergence time, it is evident that the inclusion of all the interfaces to the learning approach, made possible by the implementation of the service discovery over *Uhura*-enabled robots, helps in achieving a more complete observation of the environment. In the experiment, in fact, we obtained an accuracy of the model which is 8% higher than the performance achieved from the single interface scenarios.

Chapter 7

Conclusion and Future Work

This thesis aimed to explore the networking capabilities of WSNs and investigate how the integration of UAVs can enhance their performance. The research was divided into three main objectives: (1) Ground Wireless Mesh Sensor Networks, (2) Aerial Wireless Mesh Sensor Networks, and (3) Ground/Aerial WMSN integration.

The first objective investigated using Bluetooth Mesh for IoT monitoring in different environments, including outdoor and indoor settings. The results showed that Bluetooth Mesh can be a promising solution for IoT monitoring in different environments, providing reliable and efficient communication between devices.

The second objective focused on deploying aerial nodes to maximize the effectiveness of the data collection in terms of ground nodes connected and QoS of the UAV-to-UAV and UAV-to-Sink links while maintaining the aerial mesh connectivity. The results showed that UAVs can provide a valuable solution for data collection in WSNs, particularly in SHM applications, by providing extended communication range and reducing energy consumption.

The third objective was to investigate hybrid scenarios with air-to-ground communication links between the two network meshes. A major contribution of the thesis was the design and implementation of the *Uhura* framework, which enables Hybrid Wireless Mesh Sensor Networks and abstracts and handles multiple M2M communication stacks. The results showed that the *Uhura* framework can effectively handle multiple M2M communication stacks, providing a promising solution for integrating UAVs in WSNs.

In conclusion, the results of this thesis showed that the integration of UAVs in WSNs can enhance the performance of WSNs and provide a valuable solution for data collection in SHM applications. The *Uhura* framework can effectively handle multiple M2M communication stacks, providing a promising solution for integrating UAVs in WSNs.

7.1 Future Works

The work presented in this thesis provides a foundation for further research in the field of UAV-aided WSNs. Possible future studies that can be pursued to build upon this work include the following:

- Investigating the application of Dedicated Short-Range Communications (DSRC) for intra-swarm communication in UAV-aided WSNs and evaluating its performance in combination with other long-range wireless technologies, such as FANETs.
- Utilizing advanced machine learning algorithms, such as Multi-Agent Deep Q-Learning, for distributed SDN to improve network performance, adaptability, and decision-making of the Uhura's QoS Module.
- Exploring the integration of popular FANET protocols, such as AODV and OLSR, for UAV-aided WSNs and evaluating their impact on network performance, scalability, and robustness.
- Conducting extensive simulations and experiments under different conditions and scenarios, such as varying node density, mobility, and environmental factors, to evaluate the proposed framework and identify areas for improvement.

These research directions have the potential to significantly expand upon the work presented in this thesis and contribute to the advancement of UAV-aided WSNs in various application domains.

References

[1] Abhishek Khanna and Sanmeet Kaur. “Internet of things (IoT), applications and challenges: a comprehensive review”. In: *Wireless Personal Communications* 114 (2020), pp. 1687–1762.

[2] Karan Bajaj, Bhisham Sharma, and Raman Singh. “Integration of WSN with IoT applications: a vision, architecture, and future challenges”. In: *Integration of WSN and IoT for Smart Cities* (2020), pp. 79–102.

[3] Zhanserik Nurlan et al. “Wireless sensor network as a mesh: Vision and challenges”. In: *IEEE Access* 10 (2021), pp. 46–67.

[4] Adam B Noel et al. “Structural health monitoring using wireless sensor networks: A comprehensive survey”. In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1403–1423.

[5] Rezoan Ahmed Nazib and Sangman Moh. “Energy-efficient and fast data collection in UAV-aided wireless sensor networks for hilly terrains”. In: *IEEE Access* 9 (2021), pp. 23168–23190.

[6] Bin Liu and Hongbo Zhu. “Energy-effective data gathering for UAV-aided wireless sensor networks”. In: *Sensors* 19.11 (2019), p. 2506.

[7] Federico Montori et al. “Machine-to-machine wireless communication technologies for the Internet of Things: Taxonomy, comparison and open issues”. In: *Pervasive and Mobile Computing* 50 (2018), pp. 56–81. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2018.08.002.

[8] *Bluetooth Mesh: Recent Enhancements*. Bluetooth Special Interest Group. Jan. 29, 2023. URL: https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/.

[9] Sukun Kim et al. “Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks”. In: *2007 6th International Symposium on Information Processing in Sensor Networks*. 2007, pp. 254–263. DOI: 10.1109/IPSN.2007.4379685.

[10] Matteo Ceriotti et al. “Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment”. In: *2009 International Conference on Information Processing in Sensor Networks*. 2009, pp. 277–288.

[11] Jinping Ou and Hui Li. “Structural Health Monitoring in mainland China: Review and Future Trends”. In: *Structural Health Monitoring* 9.3 (2010), pp. 219–231. DOI: 10.1177/1475921710365269. eprint: https://doi.org/10.1177/1475921710365269. URL: https://doi.org/10.1177/1475921710365269.

[12] Ping Wang et al. “Investigation of Wireless Sensor Networks for Structural Health Monitoring”. In: *Journal of Sensors* 2012 (May 2012). DOI: 10.1155/2012/156329.

[13] Roger de Reus, Jens Ole Gulløv, and Patrick R Scheeper. "Fabrication and characterization of a piezoelectric accelerometer". In: *Journal of Micromechanics and Microengineering* 9.2 (June 1999), p. 123. DOI: 10.1088/0960-1317/9/2/005. URL: https://dx.doi.org/10.1088/0960-1317/9/2/005.

[14] Move Srl. *Wireless Accelerometer SHM*. Accessed Dec 26, 2022. https://www.movesolutions.it/accelerometershm/. n.d.

[15] Kin tak Lau et al. "Strain monitoring in composite-strengthened concrete structures using optical fibre sensors". In: *Composites Part B: Engineering* 32 (Dec. 2001), pp. 33–45. DOI: 10.1016/S1359-8368(00)00044-5.

[16] AS Fiorillo, CD Critello, and SA Pullano. "Theory, technology and applications of piezoresistive sensors: A review". In: *Sensors and Actuators A: Physical* 281 (2018), pp. 156–175.

[17] Zheng Liu and Yehuda Kleiner. "State-of-the-art review of technologies for pipe structural health monitoring". In: *IEEE Sensors Journal* 12.6 (2012), pp. 1987–1992.

[18] Steven D Glaser et al. "Sensor technology innovation for the advancement of structural health monitoring: a strategic program of US-China research for the next decade". In: *Smart Structures and Systems* 3.2 (2007), pp. 221–244.

[19] Bart Peeters and CE Ventura. "Comparative study of modal analysis techniques for bridge dynamic characteristics". In: *Mechanical Systems and Signal Processing* 17.5 (2003), pp. 965–988.

[20] H Zheng and A Mita. "Two-stage damage diagnosis based on the distance between ARMA models and pre-whitening filters". In: *Smart Materials and Structures* 16.5 (2007), p. 1829.

[21] Saurabh Bisht. "Methods for structural health monitoring and damage detection of civil and mechanical systems". PhD thesis. Virginia Tech, 2005.

[22] Jeong-Tae Kim et al. "Damage identification in beam-type structures: frequency-based method vs mode-shape-based method". In: *Engineering structures* 25.1 (2003), pp. 57–67.

[23] AK Pandey and M Biswas. "Damage detection in structures using changes in flexibility". In: *Journal of sound and vibration* 169.1 (1994), pp. 3–17.

[24] AK Pandey and M Biswas. "Experimental verification of flexibility difference method for locating damage in structures". In: *Journal of sound and vibration* 184.2 (1995), pp. 311–328.

[25] Hien HoThu and Akira Mita. "Damage detection method using support vector machine and first three natural frequencies for shear structures". In: (2013).

[26] Feng Wang and Jiangchuan Liu. "Networked wireless sensor data collection: issues, challenges, and approaches". In: *IEEE Communications Surveys & Tutorials* 13.4 (2010), pp. 673–687.

[27] Ning Xu et al. "A Wireless Sensor Network For Structural Monitoring". In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. SenSys '04. Baltimore, MD, USA: Association for Computing Machinery, 2004, pp. 13–24. ISBN: 1581138792. DOI: 10.1145/1031495.1031498. URL: https://doi.org/10.1145/1031495.1031498.

[28] Gregory Hackmann et al. “A holistic approach to decentralized structural damage localization using wireless sensor networks”. In: *Computer Communications* 36.1 (2012), pp. 29–41. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2012.01.010. URL: https://www.sciencedirect.com/science/article/pii/S0140366412000114.

[29] Igor Leão dos Santos et al. “A localized algorithm for Structural Health Monitoring using wireless sensor networks”. In: *Information Fusion* 15 (2014). Special Issue: Resource Constrained Networks, pp. 114–129. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2012.02.002. URL: https://www.sciencedirect.com/science/article/pii/S1566253512000152.

[30] Feilong Tang et al. “Wireless mesh sensor networks in pervasive environment: a reliable architecture and routing protocol”. In: *2007 International Conference on Parallel Processing Workshops (ICPPW 2007)*. IEEE. 2007, pp. 72–72.

[31] József Kopják and Gergely Sebestyén. “Comparison of data collecting methods in wireless mesh sensor networks”. In: *2018 IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE. 2018, pp. 000155–000160.

[32] Djohara Benyamina, Abdelhakim Hafid, and Michel Gendreau. “Wireless mesh networks design—A survey”. In: *IEEE Communications surveys & tutorials* 14.2 (2011), pp. 299–310.

[33] J Rejina Parvin. “An overview of wireless mesh networks”. In: *Wireless Mesh Networks-Security, Architectures and Protocols* (2019).

[34] Rattan Deep Aneja, Amit Kumar Bindal, and Shakti Kumar. “WMN routing: state of the art survey”. In: *Int J Manage Technol Eng IX (Xi)* (2019), pp. 111–115.

[35] Vinicius CM Borges, Marilia Curado, and Edmundo Monteiro. “Cross-layer routing metrics for mesh networks: Current status and research directions”. In: *Computer Communications* 34.6 (2011), pp. 681–703.

[36] Alok Kumar Gupta and Rahul Johari. “IOT based electrical device surveillance and control system”. In: *2019 4th international conference on internet of things: Smart innovation and usages (IoT-SIU)*. IEEE. 2019, pp. 1–5.

[37] Anas Dawod et al. “Advancements towards global IoT device discovery and integration”. In: *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE. 2019, pp. 147–155.

[38] Seong-eun Yoo. “A wireless sensor network-based portable vehicle detector evaluation system”. In: *Sensors* 13.1 (2013), pp. 1160–1182.

[39] Hongguo Zhang, Peihao Nian, and Yaqi Chen. “Intelligent manufacturing: The core leads industrial change in the future”. In: *2017 4th International Conference on Industrial Economics System and Industrial Security Engineering (IEIS)*. IEEE. 2017, pp. 1–5.

[40] C Muthu Ramya, M Shanmugaraj, and R Prabakaran. “Study on ZigBee technology”. In: *2011 3rd international conference on electronics computer technology*. Vol. 6. IEEE. 2011, pp. 297–301.

[41] *Digimesh Networking Guide*. Libelium. Jan. 29, 2023. URL: https://development.libelium.com/digimesh-networking-guide.

[42] *Introducing Bluetooth Mesh Networking*. Bluetooth Special Interest Group. Jan. 29, 2023. URL: https://www.bluetooth.com/blog/introducing-bluetooth-mesh-networking/.

[43] *The Fundamental Concepts of Bluetooth Mesh Networking: Part 2*. Bluetooth Special Interest Group. Jan. 29, 2023. URL: https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-2/?utm_campaign=mesh&utm_source=internal&utm_medium=blog&utm_content=introducing-bluetooth-mesh-networking.

[44] Ala F Khalifeh et al. "A simulation study for UAV-aided wireless sensor network utilizing ZigBee protocol". In: *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE. 2018, pp. 181–184.

[45] Miaoxin Pan et al. "UAV-aided emergency environmental monitoring in infrastructureless areas: LoRa mesh networking approach". In: *IEEE Internet of Things Journal* 9.4 (2021), pp. 2918–2932.

[46] Sabitri Poudel and Sangman Moh. "Medium access control protocols for unmanned aerial vehicle-aided wireless sensor networks: A survey". In: *IEEE Access* 7 (2019), pp. 65728–65744.

[47] Tu Dac Ho, Jingyu Park, and Shigeru Shimamoto. "Novel multiple access scheme for wireless sensor network employing unmanned aerial vehicle". In: *29th Digital Avionics Systems Conference*. IEEE. 2010, pp. 5–C.

[48] Dac-Tu Ho, Esten Ingar Grøtli, and Tor Arne Johansen. "Heuristic algorithm and cooperative relay for energy efficient data collection with a UAV and WSN". In: *2013 international conference on computing, management and telecommunications (ComManTel)*. IEEE. 2013, pp. 346–351.

[49] Dac-Tu Ho and Shigeru Shimamoto. "Highly reliable communication protocol for WSN-UAV system employing TDMA and PFS scheme". In: *2011 IEEE Globecom Workshops (Gc Wkshps)*. IEEE. 2011, pp. 1320–1324.

[50] Say Sotheara et al. "Effective data gathering protocol in WSN-UAV employing priority-based contention window adjustment scheme". In: *2014 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2014, pp. 1475–1480.

[51] Ling Wang, Hanshang Li, and Yingtao Jiang. "Adaptive-opportunistic aloha: A media access control protocol for unmanned aerial vehicle–wireless sensor network systems". In: *International Journal of Distributed Sensor Networks* 12.8 (2016), p. 1550147716662785.

[52] Suraj Suman, Sidharth Kumar, and Swades De. "UAV-Assisted RFET: A Novel Framework for Sustainable WSN". In: *IEEE Transactions on Green Communications and Networking* 3.4 (2019), pp. 1117–1131. DOI: 10.1109/TGCN.2019.2938403.

[53] Meng Hua et al. "Power-Efficient Communication in UAV-Aided Wireless Sensor Networks". In: *IEEE Communications Letters* 22.6 (2018), pp. 1264–1267. DOI: 10.1109/LCOMM.2018.2822700.

[54] Rezoan Ahmed Nazib and Sangman Moh. "Energy-Efficient and Fast Data Collection in UAV-Aided Wireless Sensor Networks for Hilly Terrains". In: *IEEE Access* 9 (2021), pp. 23168–23190. DOI: 10.1109/ACCESS.2021.3056701.

[55] Bin Liu and Hongbo Zhu. "Energy-Effective Data Gathering for UAV-Aided Wireless Sensor Networks". In: *Sensors* 19.11 (2019). ISSN: 1424-8220. DOI: 10.3390/s19112506. URL: https://www.mdpi.com/1424-8220/19/11/2506.

[56] Shuhang Liu et al. "Performance Analysis of UAVs Assisted Data Collection in Wireless Sensor Network". In: *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*. 2018, pp. 1–5. DOI: 10.1109/VTCSpring.2018.8417673.

[57] Juan Liu et al. "UAV-Aided Data Collection for Information Freshness in Wireless Sensor Networks". In: *IEEE Transactions on Wireless Communications* 20.4 (2021), pp. 2368–2382. DOI: 10.1109/TWC.2020.3041750.

[58] Chixiong Mao, Juan Liu, and Lingfu Xie. "Multi-UAV aided data collection for age minimization in wireless sensor networks". In: *2020 International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE. 2020, pp. 80–85.

[59] Khalid Haseeb et al. "An energy efficient and secure IoT-based WSN framework: An application to smart agriculture". In: *Sensors* 20.7 (2020), p. 2081.

[60] *Bluetooth SIG. Mesh Profile Specification Version 1.0.1, Revision Date: 2019-01-21*.

[61] Luca Sciullo, Angelo Trotta, and Marco Di Felice. "Design and performance evaluation of a LoRa-based mobile emergency management system (LOCATE)". In: *Ad Hoc Networks* 96 (2020), p. 101993. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2019.101993.

[62] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. "Low Power Wide Area Networks: An Overview". In: *IEEE Communications Surveys Tutorials* 19.2 (2017), pp. 855–873. DOI: 10.1109/COMST.2017.2652320.

[63] Emanuele Toscano and Lucia Lo Bello. "Comparative assessments of IEEE 802.15.4/ZigBee and 6LoWPAN for low-power industrial WSNs in realistic scenarios". In: *2012 9th IEEE International Workshop on Factory Communication Systems*. 2012, pp. 115–124. DOI: 10.1109/WFCS.2012.6242553.

[64] Andrea Lacava et al. "Demo Abstract: BE-Mesh: Bluetooth Low Energy Mesh Networking". In: *INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019* (2019), pp. 989–990. DOI: 10.1109/INFCOMW.2019.8845084.

[65] Maria Fazio, Antonio Celesti, and Massimo Villari. "Improving Proximity Detection of Mesh Beacons at the Edge for Indoor and Outdoor Navigation". In: *IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 2020, pp. 1–6. DOI: 10.1109/CAMAD50429.2020.9209253.

[66] *ESP-BLE-MESH framework*. URL: https://www.espressif.com/en/products/sdks/esp-idf/esp-ble-mesh.

[67] Yuri Murillo et al. "SDN on BLE: Controlling Resource Constrained Mesh Networks". In: *IEEE International Conference on Communications (ICC)*. 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761066.

[68] Hernández-Solana, ángela and Pérez-Díaz-De-Cerio, David and García-Lozano, Mario and Bardají, Antonio Valdovinos and Valenzuela, José-Luis. "Bluetooth Mesh Analysis, Issues, and Challenges". In: *IEEE Access* 8 (2020), pp. 53784–53800. DOI: 10.1109/ACCESS.2020.2980795.

[69] Raul Rondon et al. “Understanding the Performance of Bluetooth Mesh: Reliability, Delay, and Scalability Analysis”. In: *IEEE Internet of Things Journal* 7 (2020), pp. 2089–2101. ISSN: 23274662. DOI: 10.1109/JIOT.2019.2960248. eprint: 1910.03345.

[70] Philipp Zenker et al. “Evaluation of BLE Mesh capabilities: A case study based on CSRMesh”. In: *International Conference on Ubiquitous and Future Networks, ICUFN* 2016-Augus (2016), pp. 790–795. ISSN: 21658536. DOI: 10.1109/ICUFN.2016.7537146.

[71] Stefano Traini et al. “Practical Indoor Localization via Smartphone Sensor Data Fusion Techniques: A Performance Study”. In: *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2019, pp. 1–7. DOI: 10.1109/CCNC.2019.8651859.

[72] Marco Di Felice et al. “Self-organizing aerial mesh networks for emergency communication”. In: *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*. IEEE. 2014, pp. 1631–1636.

[73] Akram Al-Hourani, Sithamparanathan Kandeepan, and Simon Lardner. “Optimal LAP altitude for maximum coverage”. In: *IEEE Wireless Communications Letters* 3.6 (2014), pp. 569–572.

[74] Kurt Derr and Milos Manic. “Extended virtual spring mesh (EVSM): The distributed self-organizing mobile ad hoc network for area exploration”. In: *IEEE Transactions on Industrial Electronics* 58.12 (2011), pp. 5424–5437.

[75] Angelo Trotta et al. “Joint coverage, connectivity, and charging strategies for distributed UAV networks”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 883–900.

[76] Haitao Zhao et al. “Deployment algorithms for UAV airborne networks toward on-demand coverage”. In: *IEEE Journal on Selected Areas in Communications* 36.9 (2018), pp. 2015–2031.

[77] Vitor Dias Da Silva. *Mechanics and strength of materials*. Springer Science & Business Media, 2005.

[78] Andrea Zanella. “Best practice in RSS measurements and ranging”. In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2662–2686.

[79] Ho Jeong Na and Sang-Jo Yoo. “PSO-based dynamic UAV positioning algorithm for sensing information acquisition in wireless sensor networks”. In: *IEEE Access* 7 (2019), pp. 77499–77513.

[80] Reza Shakeri et al. “Design Challenges of Multi-UAV Systems in Cyber-Physical Applications: A Comprehensive Survey and Future Directions”. In: *IEEE Communications Surveys Tutorials* 21.4 (2019), pp. 3340–3385. DOI: 10.1109/COMST.2019.2924143.

[81] Jay Peters. *Genesis drone show used a record-breaking 3,281 drones*. Apr. 2021. URL: https://www.theverge.com/2021/4/4/22367182/hyundai-genesis-drone-show-guinness-world-record-china.

[82] Mario Coppola et al. “A Survey on Swarming With Micro Air Vehicles: Fundamental Challenges and Constraints”. In: *Frontiers in Robotics and AI* 7 (2020), p. 18. ISSN: 2296-9144.

[83] Dario Albani, Daniele Nardi, and Vito Trianni. “Field coverage and weed mapping by UAV swarms”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee. 2017, pp. 4319–4325.

[84] D. Cokuslu and K. Erciyes. “A flooding based routing algorithm for mobile ad hoc networks”. In: *2008 IEEE 16th Signal Processing, Communication and Applications Conference*. IEEE. 2008, pp. 1–5.

[85] Y. Lu et al. “Energy-efficient content retrieval in mobile cloud”. In: *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. ACM. 2013, pp. 21–26.

[86] Michael Baddeley et al. “Atomic-SDN: Is synchronous flooding the solution to software-defined networking in IoT?” In: *IEEE Access* 7 (2019), pp. 96019–96034.

[87] J. S. Mertens et al. “SDN-(UAV)ISE: Applying Software Defined Networking to Wireless Sensor Networks with Data Mules”. In: *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2020, pp. 323–328. DOI: 10.1109/WoWMoM49955.2020.00061.

[88] H.-A. Jacobsen and B.J. Kramer. “Modeling interface definition language extensions”. In: *Proceedings 37th International Conference on Technology of Object-Oriented Languages and Systems. TOOLS-Pacific 2000*. 2000, pp. 242–252. DOI: 10.1109/TOOLS.2000.891373.

[89] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[90] Waldemar Quevedo. “Introduction to NATS”. In: *Practical NATS: From Beginner to Pro*. Berkeley, CA: Apress, 2018, pp. 1–18. ISBN: 978-1-4842-3570-6. DOI: 10.1007/978-1-4842-3570-6_1. URL: https://doi.org/10.1007/978-1-4842-3570-6_1.

[91] Stefan Tilkov and Steve Vinoski. “Node.js: Using JavaScript to Build High-Performance Network Programs”. In: *IEEE Internet Computing* 14.6 (2010), pp. 80–83. DOI: 10.1109/MIC.2010.145.

[92] Leonardo Montecchiari et al. “Bluetooth Mesh Technology for the Joint Monitoring of Indoor Environments and Mobile Device Localization: A Performance Study”. In: *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2022, pp. 193–199.

[93] Tomas Baca et al. “The MRS UAV system: pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles”. In: *Journal of Intelligent & Robotic Systems* 102.1 (2021), pp. 1–28.

[94] Ala Al-Fuqaha et al. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376.

[95] Zhiwen Hu et al. “UAV Aided Aerial-Ground IoT for Air Quality Sensing in Smart City: Architecture, Technologies, and Implementation”. In: *IEEE Network* 33.2 (2019), pp. 14–22. DOI: 10.1109/MNET.2019.1800214.

[96] Wen Xie and Xiangyu Bai. “Research on Data Collection Mechanism of Wireless Sensor Network Based on UAV”. In: *2021 2nd International Conference on Computing, Networks and Internet of Things*. CNIOT ’21. Beijing, China: Association for Computing Machinery, 2021. ISBN: 9781450389693.

[97] Botao Zhu et al. “UAV Trajectory Planning in Wireless Sensor Networks for Energy Consumption Minimization by Deep Reinforcement Learning”. In: *IEEE Transactions on Vehicular Technology* 70.9 (2021), pp. 9540–9554. DOI: 10.1109/TVT.2021.3102161.

[98] Sabitri Poudel and Sangman Moh. “Medium Access Control Protocols for Unmanned Aerial Vehicle-Aided Wireless Sensor Networks: A Survey”. In: *IEEE Access* 7 (2019), pp. 65728–65744. DOI: 10.1109/ACCESS.2019.2917948.

[99] Angelo Trotta et al. “BEE-DRONES: Ultra low-power monitoring systems based on unmanned aerial vehicles and wake-up radio ground sensors”. In: *Computer Networks* 180 (2020), p. 107425. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2020.107425. URL: https://www.sciencedirect.com/science/article/pii/S1389128620311142.

[100] Jiaxun Li et al. “Joint Optimization on Trajectory, Altitude, Velocity, and Link Scheduling for Minimum Mission Time in UAV-Aided Data Collection”. In: *IEEE Internet of Things Journal* 7.2 (2020), pp. 1464–1475. DOI: 10.1109/JIOT.2019.2955732.

[101] Aditi Zear and Virender Ranga. “UAVs assisted Network Partition Detection and Connectivity Restoration in Wireless Sensor and Actor Networks”. In: *Ad Hoc Networks* 130 (2022), p. 102823. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2022.102823.

[102] Zhenyu Kang, Changsheng You, and Rui Zhang. “Placement Learning for Multi-UAV Relaying: A Gibbs Sampling Approach”. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9149409.

[103] R. Sugihara and R. K. Gupta. “Optimizing Energy-Latency Trade-Off in Sensor Networks with Controlled Mobility”. In: *IEEE INFOCOM 2009*. 2009, pp. 2566–2570. DOI: 10.1109/INFCOM.2009.5062188.

[104] Federico Montori et al. “Machine-to-machine wireless communication technologies for the Internet of Things: Taxonomy, comparison and open issues”. In: *Pervasive Mob. Comput.* 50 (2018), pp. 56–81.

[105] Tianxiang Li et al. “Distributed Dataset Synchronization in Disruptive Networks”. In: *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems*. 2019, pp. 428–437. DOI: 10.1109/MASS.2019.00057.

[106] Leonardo Montecchiari et al. “Distributed Service Discovery over Heterogeneous Robotic System-of-Systems”. In: *submitted for publication to IEEE International Conference on Communications (ICC)*. 2023.

[107] Leonardo Montecchiari et al. “Autonomic Faulty Node Replacement in UAV-Assisted Wireless Sensor Networks: a Test-Bed”. In: *accepted in IEEE Consumer Communications & Networking Conference (CCNC)*. 2023.

[108] Leonardo Montecchiari et al. “Uhura: a Software Framework for Swarm Management in Multi-Radio Robotic Networks”. In: *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE. 2022, pp. 244–251.

[109] Angelo Trotta et al. “A GPS-free flocking model for aerial mesh deployments in disaster-recovery scenarios”. In: *IEEE Access* 8 (2020), pp. 91558–91573.

Acknowledgements

September 2011
- Victor started crying immediately after the train left the Civitanova Marche-Montegranaro station. At the same time, Ennio was winking at me. Hero.

October 2011
- "Ce vidimu su li..." quote.

2011-2014
- The Padadelli team milled exams like no tomorrow.

October 2014
- "La Casa Del Piacere" was born.

November 2014
- "You are not my priority" comes into play for the first time.
- Phantom F12 farewell.

March 2015
- **ACHIEVEMENT** Bachelor's Degree: Hodor, An approach for disambiguation of authors of scientific articles
- Victor Cried.
- Ennio patted me on the shoulder.

September 2015
- Modal Nodes opened the gates for the first time at Via Castiglione.

October 2015
- "Mannaggia a me" quote.

July 2017
- Black Mirror Project came into existence.

February 2019
- achievement Parallel algorithms, 18, one week, good job my friend.

AROUND JUNE 2019
- First met with Giulia.

Summer 2019
- `Angel-PumaRidd-DiFelix`'s Sacrifice - `PhoeniX` was created.

October 2019
- **ACHIEVEMENT** Master's Degree: Sybelius, Design and validation of a controlled mobility platform for UAV-aided sensor networks
- Victor Cried.
- Ennio patted me on the back.

November 2019...
- Thanks to the pandemic, it was a really funny Ph.D. journey, especially in the beginning.

2023
- By the way, in the end, it seems that everything should have an ending.

Thank you, **Marco**, for being my academic guide with endless patience.
Thank you, **Angelo**, for being a top-class co-supervisor.