University of Catania

Department of Matemathics and Computer Sciences

XXXIII Ph.D. in Computer Science (International)

Federico Fausto Santoro

Communication, Elaboration and Artificial Intelligence IoT solutions for the Industry 5.0

Doctoral Thesis

University Advisor: Prof. Corrado Santoro

Enel Green Power Advisors: Dario Iuvara
Giuseppe Leotta

Academic Period 2018 - 2020

"*Never send a human to do a machine's job.*"

"Agent Smith",*Matrix.*

Abstract

The use of low-power wireless sensors and actuators with networking support in the industry has increased over the past decade. New generations of microcontrollers, new hardware for communication, and the use of standardised protocols, such as the Internet Protocol, have resulted in more possibilities for interoperability than ever before. This increasing interoperability allows sensors and actuator nodes to exchange information with large numbers of peers, which is beneficial for creating advanced, flexible and reusable systems. The implementation of these Internet of Things devices in the industry environment, allowed a production improvement, in terms of quality, safety and automation. Moreover, the companies which are researching this field made new types of machines, robots and unnamed aerial vehicles, which are useful in these production process. The main contribution of this thesis is research and solution proposes of efficient Industry Internet of Things, not just for industrial applications but also for the Internet of Things in general. The thesis deals with the three macro categories which are related to communication, privacy preservation, simulation, performance and application fo artificial intelligence in the Internet of Things devices. In the first part, the thesis focuses on new communication protocols and solutions for the Internet Of Things, introducing the design of new LoWPAN auto-maintaining protocol, an improved protocol for delay tolerant networks and an application solution for the unmanned aerial vehicles control using the fifth generation of cellular systems. The second part of the thesis introduces a new Docker technology improvement, which improves the normal building process of Docker. Furthermore, the thesis discusses a new simulator framework about unmanned aerial vehicles and their wireless communication protocols. In the third and final part of the thesis, new applications of artificial intelligence were introduced. In particular, a new architecture for human interaction with domotic devices through social channels was discussed. Besides, a new neural network model was introduced in the final part. The proposed model improves the production of solar panels through the ability to predict the resulting solar panel before its creation. Finally, at the end of each chapter, a set of experiment results and conclusions were presented.

Acknowledgements

This thesis is the result of more than two years of continuous research, development, and learning. During that time I met many people who helped me to be the researcher I am now; because without their advice this thesis would not have been possible.

I need to extend my gratitude to my supervisor, Professor Corrado Santoro, who trusted me for this Ph.D. position and invested time and effort guiding me to be on the right track. Because I have found not only a guide but also a friend in him.

I also need to manifest my gratitude to my friends, Professor Salvatore Riccobene and Professor Fabrizio Messina for their unconditional help, friendship, collaborations, and discussions during these long way.

I would like to thank all my colleagues at the University of Catania because, in one way or another, they contributed to this with a comfortable working and enjoying environment.

In particular, I would like to thank all my 200th room colleagues for being patient with me during these three years. Thanks to you for helped me growing, maturing and I can confirm that I have found a small family, thanks, Santi Orlando, Francesco Ragusa, Luca Guarnera, Marco Rosano, Filippo Milotta and Lorenzo Di Silvestro.

I also thank my friend and colleague Riolo Salvatore, with whom I had the opportunity to laugh, study and make new international experiences.

Special thanks go to Enel Green Power S.p.A. and to my advisors, Giuseppe Leotta and Iuvara Dario and company colleague Eleonora Arena, who helps me in the last period.

Moreover, thanks to Xacria S.r.l. and the employers, with which I made a lot of Cloud and Internet of Things projects, that helped me to grow in my research, in particular Filippo Randazzo who was very patient with me.

Appreciation is also expressed to all my students and undergraduates with which I worked in the last years and to Daniele Mulà that helped me with my university course.

Contents

List of Figures

List of Tables

To my mother, which always believed in me.
To my love, which has been with me in the worst times.
To my father, that can no longer be here but pushed me to take this.
I love all of you, Giuseppina, Daniela and Francesco.

Chapter 1

Introduction

The Internet of things (IoT) term is new as old and was mentioned by Kevin Ashton in 1999[1], taking up the idea of radio frequency identification (or RFID) with what was at the time the leading topic, namely the Internet. From that event, many companies predicted the dissemination of this technology[2], as the increase in the number of things interconnected between them and, also, through the Internet[3]. This increase was renowned in the last ten years[4].

Many believe that the succession of such events is mainly due to the famous laws of Moore[5] and Koomey[6]. According to Moore, the number of transistors within the chips doubles about every two years, allowing the development of more powerful machines, even the size of a single chip. Koomey explains instead that the number of computations per kilowatt-hours doubles approximately every year. Considering the two mentioned laws, it is easy to see how these also represent that machines can become smaller, also decreasing the energy required to perform the computations that the chip can perform, effectively making these more energy-efficient.

There is no real definition of the IoT, although several research groups have given their definition. In general, we can say that "the IoT is defined as a dynamic and global infrastructure, with the ability to self-manage based on interoperable communication standards and protocols"[7], more simply, "things" capable of communicating with each other or allowing users to communicate with these devices.

The IoT combines different technologies within what then becomes a semi-autonomous network of various types of devices. These are classified according to their functionality, such as sensors, communication, computation, identification or service, which are connected to the software technologies present today. Considering also that these devices can communicate through the Internet, it is easy to

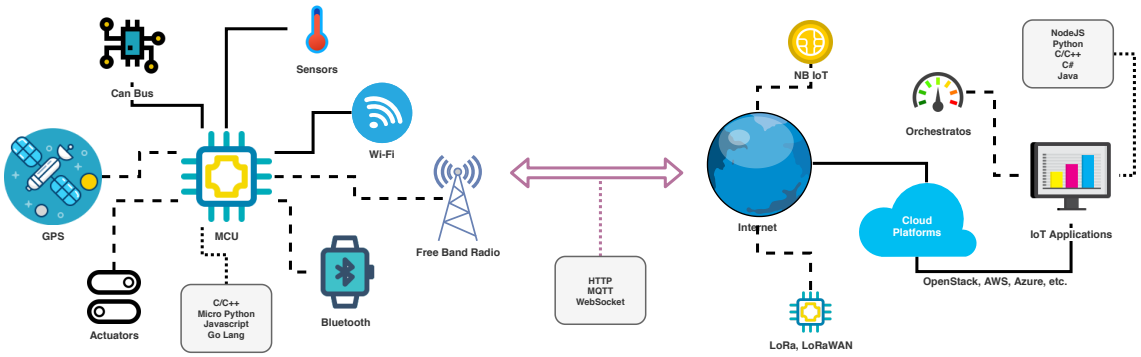


Figure 1.1: Internet of Things Architecture and technologies.

imagine an infinity of types of devices that can be designed.

An IoT device is equipped with a series of sensors that allows it to draw information from the environment that surrounds it, such as environmental temperature, humidity, light intensity, or more commonly, its physical position in an open environment via GPS. These devices are also equipped with computational capabilities which allows them to process information and may perform any type of operation (e.g. a thermostat that turns off according to the environmental temperature).

These devices are also equipped with communication modules, which allows them to communicate wirelessly, thereby using high-level messaging protocols, services and cloud infrastructure through Internet protocol (IP). Among the most frequently used messaging protocols are HTTP and MQTT protocols, which are useful when devices need to publish or share information at that moment (e.g. a sensor reading). When a realtime communication is necessary, appropriate protocols, such as Websocket, should be used.

The most important part of IoT is how these devices are connected in order to communicate. This ability is essential when you want to label an object as an IoT object. However, "the how" is less important, because the physical links and the MAC layer for these devices, can be made in different ways. The implementation of this depends on the purpose, the use and the environment. In general, there are three methods of communication within the IoT.

The first method concerns those devices that do not need to communicate through a network but communicate directly with another device. This kind of communication, also known as peer to peer (P2P), becomes very useful when two devices are close enough to be able to communicate in a more simplified way. This type

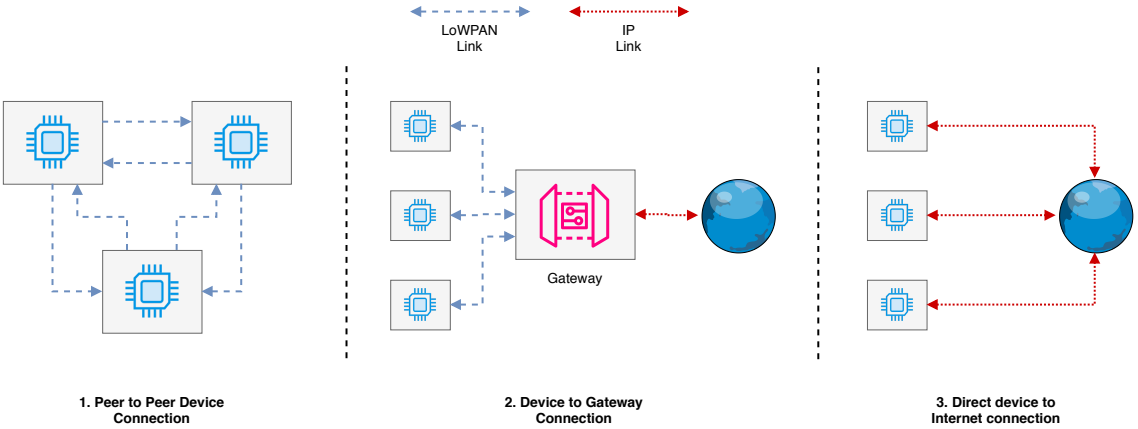


Figure 1.2: Internet of Things Communications modes.

of communication usually occurs through radio technologies, such as Bluetooth or Zigbee, the latter based on 802.15.4.[8].

Another way is to communicate through the Internet with the use of a gateway. These devices use low-energy medium/long-range radio technologies, such as 6LoWPAN or LoRa, to communicate with a common gateway, which has the same network interface. This gateway finally manages to send various messages received by the devices to possible servers via Internet Protocol (IP). A practical example is the LoRaWAN protocol.

Finally, a device may have the ability to communicate through the Internet without the aid of any gateway, for example by taking advantage of commonly used network interfaces, such as Ethernet or Wi-Fi or via the mobile data connection (3G, 4G, NB-IoT).

From this, we understand how interconnectivity is the most important feature for the IoT because the basic concept is built precisely on the ability to communicate with everything trying to get a heterogeneous system. This is also the most difficult point to reach, as there are many communication protocols, all different from each other to date. Designing and implementing new communication solutions by using more than one protocol has become a subject of research that seeks to address technical and security issues. These issues are based on the purpose, the use and the environment.

IoT applications are applicable in almost any field, thanks to the fact of low energy consumption, which makes it possible to design and develop battery-powered

devices, which can be used in environments without it. Smart homes are equipped with advanced automatic systems. Some can be scheduled for various operations or tasks, such as light management, power consumption, video and audio control, security and alarm systems, etc. These environments are usually referred to as intelligent environments which are sensitive and adaptive to modern human needs[9]. Nowadays, the objectives of a smart home are to simplify energy management and reduce unnecessary consumption[10]. This is important because energy consumption and the comfort of the occupants are key factors when talking about smart home environments[11].

One of the most striking fields of IoT, in recent years, is certainly the industrial one. We often talk about the industrial revolution and how this has changed enterprises in the last decades. The first great revolution began in 1790 in Great Britain, implementing the aid of motor machinery. This revolution changed several techniques and commodities. Among these, for example, are the replacement parts for the same machines which lead to the first mass production. Later, in the 20th century, the automobile industry revolutionised the business world again, know as the Second Industrial Revolution. The biggest novelty was the creation of the so-called assembly lines, introduced in the early 1900s. There was also a third revolution, which was represented in a more general way by the introduction of telecommunications, bringing production and sales to a global rather than just a local level.

The fourth industrial revolution[12], the one we are currently experiencing, comes from the implementation of new modern technologies, such as the IoT, called Industry IoT (IIoT)[13]. It defines a sub-category of IoT concerning technologies applied to the industrial environment. This has proven to be a key technology for the Industry 4.0, by improving and optimising production processes through the connection between machinery; the realisation of data useful for the production analysis; preventive checks on the state of production, and control of production times in general. The IIoT also represents an evolution of the IoT, allowing a device to have multiple connections at the same time and to work with a greater amount of data.

Compared to normal IoT devices, the IIoT ones need to be more resistant, as they must operate in extreme conditions. Usually, these devices are used in environments where the temperature or the risk of corrosion is high or immersed in

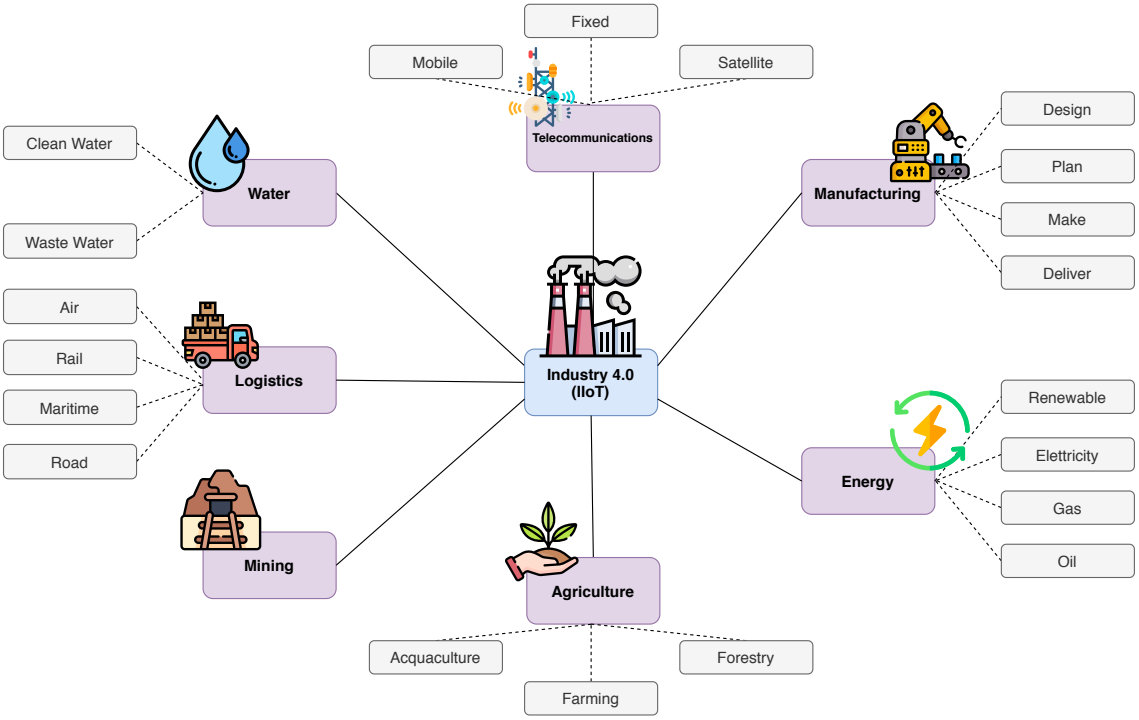


Figure 1.3: Industry 4.0 / IIoT fields.

water. In some extreme situations, where the environment has no power supply for such devices, these are equipped with special batteries which are able to withstand these types of environments. These devices have greatly improved the quality of production and productivity itself. Many of the production processes, especially those that require repetitive operations[14], are now automated by machines and robots that communicate their status and production through communication channels to a data center; data that will be processed later[15]. It is obvious to see that the industrial revolution not only brought the use of IIoT in production but also in safety requirements. Thanks to these devices it is now possible to monitor the state of the air in a given place; possible technical faults in a machine that could become dangerous to workers and the various automation processes that may prevent unfortunate accidents. All this, however, has a cost. The implementation of such devices can be expensive for a company. Furthermore, the same devices must be designed and programmed ad-hoc for the task they will have to perform. Even the design and development of such devices is not trivial. In a manufacturing environment, these devices must guarantee a certain degree of reliability and resistance to failures; and

any certifications that may eventually bring the device to the market may be costly and difficult to obtain[16].

Even though the Industry 4.0 was a clear industrial revolution along with IoT technologies, there is room for improvement[17]. Recently, there has been talk of a new fifth industrial revolution, which would bring artificial intelligence (AI) within the production process to further improve production. In reality, it may seem strange to many that the implementation of an AI within a production line can assert itself as a real revolution, as artificial intelligence is already used within these companies. Nowadays, the IIoT is based on a continuous collection of information and communication to a data center, where it will be stored. This information is then used to instruct an artificial intelligence to predict failures and malfunctions within their production chain. The goal of the fifth industrial revolution is to create devices capable of making decisions, mostly simple, within a production line. An example is an infrastructure that would simplify connections between individual devices and a centralized server, particularly when these are powered by batteries alone. The ability to make decisions in complete autonomy is the simplest way to avoid unnecessary wireless communications, reducing the energy consumption of devices and prolonging their operation.

The concept of artificial intelligence is mostly linked to machine learning techniques, which is a subcategory of AI. Machine learning can be defined as a set of methodologies that have been developed during the years, such as pattern recognition, image elaboration, data mining and deep learning. In the scientific field, machine learning is often linked to computer vision. Nevertheless, in recent years these methodologies have found application in other areas, such as data processing and IoT. Deep learning techniques, particularly neural networks, have shaken global research in recent years. The possibility of generating a neural network model and being able to use it in IoT devices would make these intelligent capable of making relatively simple decisions, with a good degree of reliability. A practical example would be a smartwatch. Depending on the wearer's heart rate, humidity and other sensors data, it can predict a possible malaise of the wearer, and contact emergency services promptly. In industry and production environment, this would lead to new infrastructures, where individual machines can, for example, understand or predict the quality of a final product, possible local failures, or energy consumption and, as

a consequence, production costs.

1.1 Problem definition

Even though IoT has evolved in recent years there are still some doubts and known problems. In IoT, data and information communications are the key to its infrastructure, but it is not designed to be secure. Even the availability of many network protocols makes it, ironically, difficult to manage. Many of these are proprietary protocols and are not free to date. They work with frequency bands that sometimes are limited in many countries around the world. This problem has made the design of several commercial and all types of open-source solutions very popular, nonetheless leading to confusion, making it difficult to understand them. Furthermore, the implementation of the same communication protocols is not trivial, and the choice of which to use depends on the application environment which often leads to bizarre solutions that are not entirely heterogeneous. A clear example is commercial products for home automation. Most of these devices work via a normal Wi-Fi connection, though others use Zigbee or other technologies (maybe proprietary) as a commercial solution, making it difficult to interface with such instruments.

Another topic worthy of note is privacy, taking as an example the same IoT devices for home automation. These exploit external services in the cloud, effectively, transferring information and voice recordings for the smooth operation of the device. The situation is worse concerning even the most modern security cameras which constantly send audio and video streams of our lives today. Even the various elaborations that face the devices are not to be underestimated. Many of these IoT devices, however powerful, have limited reasoning capabilities, leading to long processing times sometimes blocking the device. Moreover, the problem could also exist in resources in general, such as storage space, where information or neural models are stored.

Finally, there are still several difficulties to be overcome. These are current research topics that need to be studied and resolved. An example is the indoor localisation, which is useful inside of a company or a production chain, especially if they want to automate the movements of logistic robots. Unmanned Aerial Vehicle

(UAV) is also a topic of interest, increasingly used in the industrial field, as well as in logistics.

1.2 Thesis Scope

The Internet of Things is a vast area of research that has evolved steadily over the last two decades. As a result, it represents a huge area of study that involves a combination of many disciplines: hardware, security, communication, cloud computing, big data, deep learning, etc. This multi-disciplinary nature of IoT requires the constant collaboration of researchers with different research curricula. There are so many problems in this area today, and no one can deal with them all. During the period of my PhD, I decided, together with my associates and my academic adviser, to focus in particular on three specific areas of IoT and its current research issues or topics which are mostly dedicated to the industrial field but applicable to any context.

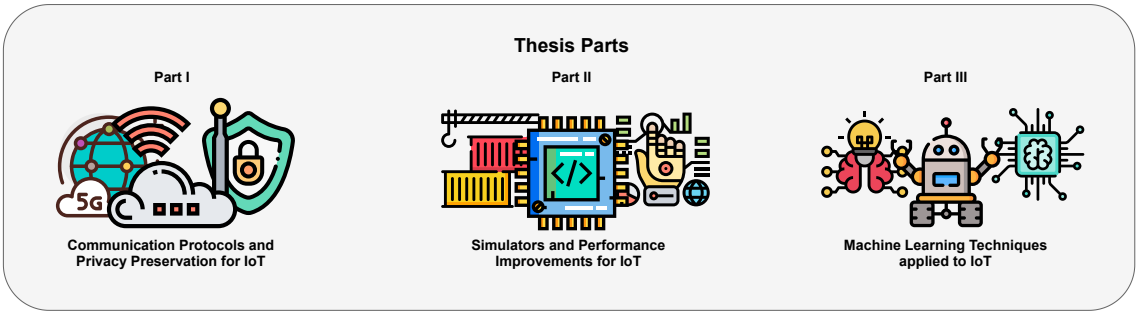


Figure 1.4: IoT Arguments of this Thesis.

This thesis focuses, in detail on the study of the feasibility and design of new solutions for IoT. It also proposes and analyses the problems and solutions considering three macro areas of IoT, such as Communication Protocols and Privacy Preservation, Simulators and Performance Improvements and Machine Learning applied in the IoT environment. All aspects of this research are not strictly focused on specific devices, but find applications in a wide range of these, such as UAVs, embedded devices and microcontrollers. This thesis, therefore, represents a set of improvements regarding IoT, such as scalability, information processing, communication, privacy, interoperability and energy efficiency.

1.3 Thesis Outline

The thesis consists of three parts, each organised with a macro area. The first part presents research and solutions regarding the applicability of the new fifth generation of mobile communication (5G), especially applied to UAV systems for coordinated flight control. It also introduces an improvement applicable to any delay-tolerant network, improving privacy preservation of information exchange between nodes. Finally, a LoWPAN protocol, based on the 802.15.4 standard, capable of auto-adapting to any situation or failure, will be presented.

In the second part, two innovative solutions will be presented, regarding simulation and embedded computing. The first solution is a complete simulator of UAV systems, capable of simulating through the integration of network-simulator 3 (NS-3) wireless communication systems realistically, allowing you to install and use the simulator in a distributed way. The second solution concerns an approach to building Docker images, which by default, which must ensure privacy between the various containers and would generate used storage space unnecessarily. The new approach, called Wale (assonance with the word Whale), stands above Docker without altering the internal operation, allowing you to save storage space and managing to share packages and libraries between containers, without denying privacy between them.

Finally, in the third part, other solutions related to machine learning techniques applied to the IoT environment will be discussed. The first is based on the development of a social assistant, called Fabulos, which allows users to interface with their home automation devices through social channels. This software tool defines a Social Network Interface and a BDI Interface Engine which is a system with artificial intelligence, based on rules, that implements the logic of interactions and the various tasks to be performed. Lastly, a neural network model will be introduced which can predict the maximum power production of a solar module. A set of solar cells is given as input to that neural network. The model was made using a dataset of real data, produced during production tests, thereby making it ready for use in production for tests and simulations. The proposed model reached an excellent level of accuracy during the test phases and has been implemented within a cross-platform software that allows predictions and simulations, even via the network.

Publications

1. **Integrating Heterogeneous Tools for Physical Simulation of multi-Unmanned Aerial Vehicles**[18]
   Authors: *Fabio D'Urso, Corrado Santoro, Federico Fausto Santoro.*
   Status: *Published in 19th Workshop on Agents conference (WOA2018).*

2. **The Tactile Internet for the flight control of UAV flocks**[19]
   Authors: *Fabio D'Urso, Christian Grasso, Corrado Santoro, Federico Fausto Santoro, Giovanni Schembra.*
   Status: *Published in 4th IEEE Conference on Network Softwarization and Workshops (NetSoft).*

3. **Wale: a Dockerfile-based approach to deduplicate shared libraries in Docker containers**[20]
   Authors: *Fabio D'Urso, Fabrizio Messina, Corrado Santoro, Federico Fausto Santoro.*
   Status: *Published in IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech).*

4. **An integrated framework for the realistic simulation of multi-UAV applications**[21]
   Authors: *Fabio D'Urso, Corrado Santoro, Federico Fausto Santoro.*
   Status: *Published in Computers & Electrical Engineering* ***Journal****.*

5. **Meaning Extraction in a Domotic Assistant Agent Interacting by means of Natural Language**[22]
   Authors: *Carmelo F. Longo, Fabrizio Messina, Corrado Santoro, Federico Fausto Santoro.*
   Status: *Published in IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE).*

6. **Design of Self-organizing Protocol for LoWPAN Networks**[23]
   Authors: *Matteo Buffa, Fabrizio Messina, Corrado Santoro, Federico Fausto Santoro.*
   Status: *Published in 12th Internet and Distributed Computing Systems International Conference (IDCS 2019).*

7. **Wale: A solution to share libraries in Docker containers**[24]
   Authors: *Fabio D'Urso, Corrado Santoro, Federico Fausto Santoro.*
   Status: *Published in Future Generation Computer Systems* ***Journal****.*

Part I

Communication Protocols and Privacy Preservation for IoT

Chapter 2

Flight control of UAV flocks through fifth Generation Mobile Network

2.1 Introduction

Unmanned Aerial Vehicles (UAV), also known as "drones", are nowadays devices widely available, and that can be bought off-the-shelf. Small UAVs can be easily obtained from sellers, and their employment ranges from toys to professional applications. A wide range of monitoring applications is now made possible with reasonable costs thanks to the availability of such small professional UAVs that can be equipped with cameras or other kinds of sensing devices.

However, one of the main drawbacks of such UAVs is their autonomy that, with the technology currently used for high capacity batteries, allows a flight time of only 15-20 minutes. This limitation impedes long-term missions and, as consequence, restricts the size of the areas to monitor. To avoid such a problem, an approach proposed by several researchers is the adoption of a *flock of UAVs* [25, 26, 27] that, by flying in a proper formation, can—in principle—cover an area that is n times (with n the number of drones) larger than the area covered by a single drone.

State-of-the-art solutions exploit *decentralized* approaches: each UAV is equipped with a proper hardware-software infrastructure able not only to ensure flight stabilization and control, but also to interact with other UAVs (by means of a communication technology) in order to plan all together the best path for the mission to perform. The result is an *emerging behavior* that drives the flock towards the

coverage of the area to be monitored. Using a distributed/decentralized architecture is the key to ensure also load balancing and fault tolerance. Moreover, thanks to the autonomy, should a UAV fail the other UAVs can detect the event and adopt proper countermeasures.

As it is well known, any decentralized architecture that is asked to support an emerging behavior needs a proper *communication infrastructure*, otherwise entities cannot self-organize. Wireless (ad-hoc) networks can, in this case, serve for the purpose, but they suffer of two main drawbacks:

- *Limited Range.* Wireless devices used in the UAVs (as in any wireless sensor network) have a limited range, so a complete coverage of the whole flock could not always be possible (above all when the flock is large). In this case, routing protocols must be adopted thus increasing the communication latency.

- *Medium Access.* A well-known key problem of wireless ad-hoc networks is the medium access control. Even if specific protocols are used, the probability of packet collisions on the same shared transmission medium is always present, thus constituting another factor that affects bandwidth and latency.

A solution to the above issues is provided by the incoming fifth generation (5G) of cellular systems [28, 29, 30, 31, 32, 33]. One of the main peculiarities of 5G technology is the introduction of Multi-Access Edge Computing (MEC) [34, 35], an ETSI proposal that leverages on network softwarization paradigms like Software Defined Networks (SDN) [36] and Network Functions Virtualization (NFV) [37]. Leveraging on the above paradigms, a Telco operator is able to offer application developers and content providers cloud-computing capabilities at the edge of the network to achieve ultra-low latency and high bandwidth, as well as real-time access to radio network information.

Another key element introduced with 5G and complementary to MEC is the concept of *network slicing* [38, 39], defined by the Next Generation Mobile Network Alliance (NGMN) as an independent virtualized end-to-end network allowing operators to run different deployments based on different architectures in parallel. Specifically, the term network slice refers to an instance of such a logical network using network and application function chains for delivering services to a given group of devices. An exemplary application for network slicing is the “Tactile Internet”,

with the purpose of achieving interactions between a human or machine and physical objects in timescales of 1 ms. Extremely high reliability and security are additional requirements, nonetheless the mission-critical characteristics associated with its applications.

In this section, we propose a software architecture based on a 5G communication infrastructure to control a flock of UAVs in their monitoring mission. The architecture is composed of three main modules: *flight control*, which is the flight stack needed to control stability, speeds and waypoints, the *flock control*, which is in charge of deciding the UAV movements to maintain a given shape of the flock, and the *mission control*, which is in charge of controlling the mission by sending proper commands to the UAVs. The above modules usually run on board the UAVs because of the strict interaction needs with the controlled UAVs. Instead, in this chapter, by taking advantage of ultra-low latencies guaranteed by the 5G Tactile Internet, these modules are designed to run on the *edge of the network*, i.e. in proper servers placed either in the edge cloud deployed in the Central Office (CO) at the access point of the core network, or even in the base stations, according to the latency requirements between them and the UAVs they are controlling.

These modules also feature *replication* in base stations of adjacent cells in order to support the continuity of operations even when one or more UAVs perform the handover. Finally, some additional modules not presenting so hard requirements, can be run on remote core clouds, and this is necessary if they require very huge amounts of storage and/or computation.

2.2 Related Works

We consider, as a reference scenario, a certain area of terrain—more or less wide—to be overflown for a specific reason, e.g. aerophotogrammetry, video or IR inspection, seeding or fertilizer spreading for agriculture, etc. Here a set of UAVs is employed, with the objective of subdividing the whole area into portions, each overflown and monitored by one UAV. The way in which the area is subdivided and the parts assigned to UAVs vary on the basis of the employed approach.

Some solutions presented in the literature [40, 41, 42] adopt a centralized model in which a (centralized) entity performs an off-line (batch) processing in order to

determine area partitions, thus assigning each partition to a different UAV that will execute the overflight in autonomy. The central entity has also the task of checking that all UAVs are working and, in case of failure, can assign the area of the faulty UAV to another (or more) working UAVs.

Other proposals [43, 44, 45, 25] adopt a completely decentralized approach by avoiding any central entity, and distributing the mission algorithm to the UAVs themselves. In this case UAVs cooperate by exchanging messages and self-organize in order to *(i)* form and maintain a "flock of drones"; and *(ii)* plan the optimal path to cover the area to monitor. By continuously exchanging messages, each UAV can also detect if another UAV fails (i.e. it becomes "silent") and, altogether, re-plan the path in order to re-scan areas lost by the failed entity. This kind of solutions implies that UAVs can continuously interact to each other, therefore they need a proper communication infrastructure able to guarantee certain quality-of-service parameters.

In this chapter we consider the latter approach, i.e. a distributed solution to control a flock of drones belonging to an aerial monitoring platform, but leveraging on a 5G Tactile Internet slice as the communication and computation commodity. This slice provides support for ultra-low latency communications among all the elements constituting the monitoring platform considered in this chapter. The physical structure of the 5G network is sketched in Figure 2.1. It is hierarchical from the radio access network (RAN) to the remote Internet. The RAN is realized as a Cloud-RAN (C-RAN), that is, constituted by an additional layer of small cells into the existing macro-cell network, deployed as remote radio heads (RRHs), and connected to a cloud where baseband units (BBU) are pooled in a single geographical point for a one-to-one logical mapping.

The fixed network comprises the access and the core domains. The *Fixed Access Network* interfaces the radio link with the *Core Network*. While the core domain is commonly implemented with optical transport, the access domain uses a heterogeneous set of transport technologies. However, adopting optical transport also in the access domain is a key enabler for the Tactile Internet as it offers high capacity and small propagation delays, mandatory features to achieve the requirements imposed to the Tactile Internet.

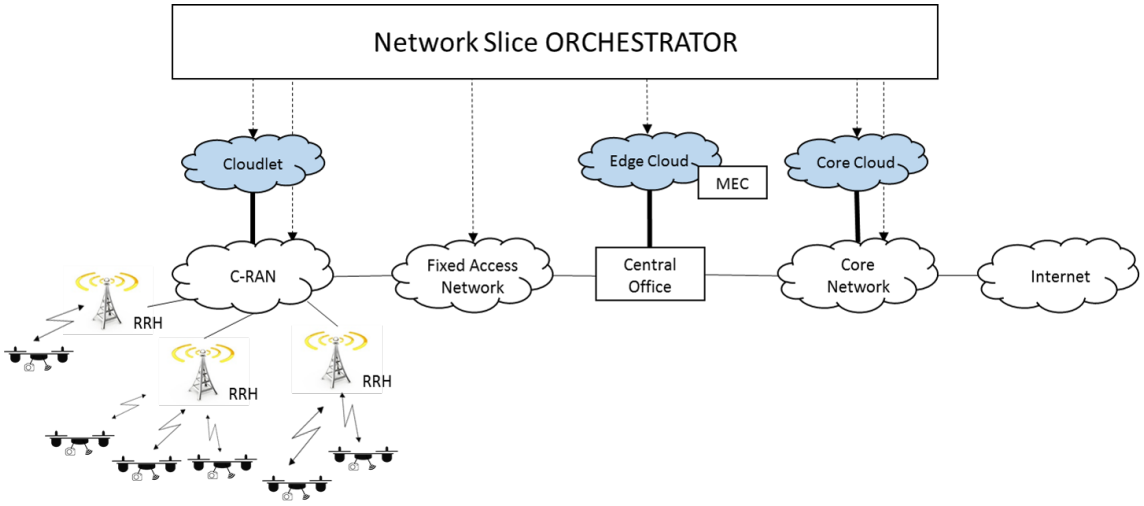


Figure 2.1: Hierarchical Cloud Architecture of a 5G system.

Distribution of physical resources, as Figure 2.1 shows, allows different deployment levels of computing resources, all open to third-party service providers to run their applications:

- The *Core Cloud*, which is a centrally located data center that hosts a large collection of processing, storage, networking, and other fundamental computing resources. On this cloud, the provider of the considered aerial video-monitoring service is allowed to deploy and run software, e.g., operating systems and applications, to realize its service. Typically, only a few core clouds are installed in a nationwide telco Operator network.

- An *Edge Cloud*, which is implemented inside an access branch of the fixed network, closer to the end user, typically deployed inside a central office (CO). This location constitutes a good trade-off between proximity with end-users and devices, and computation and storage capacities.

- A *Cloudlet*, or *Nano-Cloud*, which is a mobility-enhanced small-scale cloud data center, usually co-located with the macro cell sites. The main purpose of the cloudlet is hosting the deployment of the Tactile Support Engine where running resource-intensive and mobile applications to provide ultra-low latency services and Artificial Intelligence (AI) to mobile devices like the UAVs we are considering in this chapter.

Physical and virtual resources belonging to the underlying network infrastructure are managed and orchestrated by the *Orchestrator* entity, which is in charge of the lifecycle of all the network and application functions, their placement and their mapping on the available physical resources.

The optimal allocation of processing functions composing the aerial video-monitoring service application on the different levels of computing resources described so far will be described in the following according to the interaction requirements with the UAVs.

2.3 Control of UAV Flocks and Performance Requirements

As said so far, the best strategy to control UAV flock flight is by applying a distributed approach. However, difficulties in propagating information from each UAV to all the others belonging to the same flock due to the short-range communications, strongly limit the number of UAVs composing a flock, their maximum allowed speed and the UAV density in the flock.

The idea at the base of this chapter, that can be realized only thanks to the usage of a 5G Tactile Internet slice, is to create a digital twin of each UAV with a Virtual Drone image running on the ground as a chain Virtual Application Functions (VAF), and timely distributed among the three different cloud levels shown in Figure 2.1.

As depicted in Fig.2.2, the proposed aerial video-monitoring service application can be realized with a number of Virtual Drones, each representing a virtual image of a physical drone participating to the monitoring mission, and a set of additional elements aimed at coordinating them and integrating their work with additional facilities. Each component of the above elements is implemented as a VAF running inside an execution container like, for example, a virtual machine, a Linux Container or a programmable embedded system. VAFs chained to realize a Virtual Drone, as well as the additional VAFs *Global Video Processor*, *Historical Video DB* and *Mission Manager*, will be described in details in Section 2.3.1, while interfaces will be defined in Section 2.3.2.

2.3.1 Virtual Application Functions

UAV Flight Controller. The *UAV Flight Controller* is one of the most critical parts of the system. It provides the low-level functionalities to perform control, and stabilization of a UAV and implements the overall classical flight stack by exploiting the inertials and positioning sensors that are installed on the physical UAV. Indeed, the state-of-the-art technique employed to control a UAV is to determine its *attitude* and *position*, by computing Euler angles—*roll*, *pitch* and *yaw* and their derivatives—from the data by an Inertial Measurement Unit (IMU[1]), and the global position by means of GPS and a barometer. These data are analyzed and processed by the stabilization software that implements the control loops and, in turn, outputs data for driving propellers in order to let the UAV keep the desired target attitude and position.

Since this VAF has to interact with the physical entity, it must comply with the requirements related to the dynamics of UAV control loops that, in general, must be in the order of 500 Hz (i.e. 2 ms of maximum duration) and requires strict real-time computations otherwise the system would result uncontrollable. This is the reason why flight stacks are usually implemented by means of embedded/microcontrollered systems that, however, even if they can provide the needed real-time requirements, usually present a not so high computation power that, on the contrary, would be really needed for data processing and sensor fusion (e.g. Kalman filters) implementation.

Flock Controller. The *Flock Controller* is in charge of implementing the Flocking formation and maintenance algorithm. We consider, in particular, the flocking algorithm presented in [25, 43], that we briefly describe here[2]. The objective is to establish a flock shape that can be optimal to perform area scan; the used approach exploits a decentralized algorithm that, on the basis of the mutual positions of the various UAVs, determines the target horizontal and angular speeds to be applied to UAVs, in order to ensure that the desired flock is maintained. This process is performed by each Flock Controller using an autonomous loop that:

[1] An IMU is in general made of 3-axial gyros, accelerometers and magnetometers

[2] The reader interested in understanding the details of the algorithm can refer to the cited bibliography.

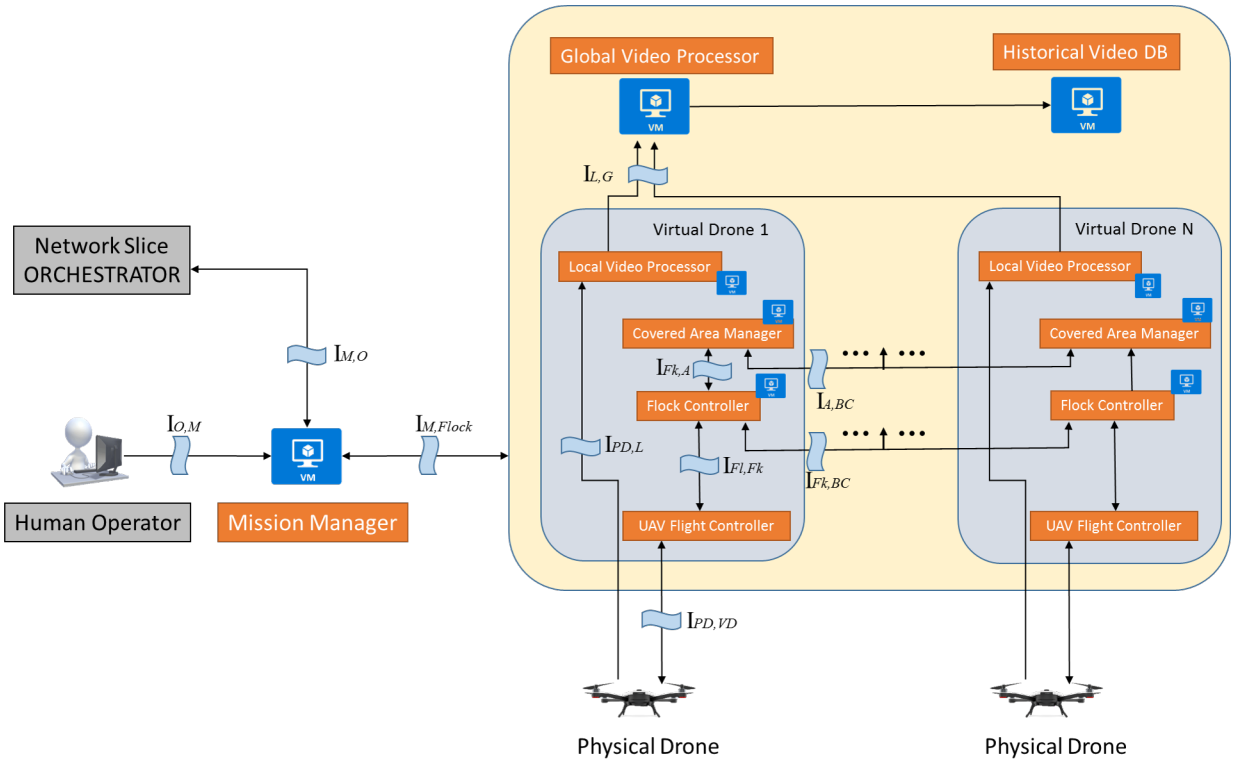


Figure 2.2: Virtual Infrastructure of the Proposed Architecture.

1. Obtains the positions of the other UAVs by contacting the relevant Flock Controllers;

2. Executes the control algorithm according to [25, 43];

3. Sends the computed target speeds to the UAV Flight Controller in order to apply them to the physical UAV.

The dynamics of execution of the control loop of Flock Controller is in the order of $\frac{1}{50}s$ while, as for the amount of data exchanged, the Flock Controller obtains GPS position (latitude, longitude, heading and altitude) from the UAV Flight Controller and the positions of the other UAVs from peer Flock Controllers; this last set of data is dependent on the number of UAVs employed in the mission.

One of the main characteristics of the flocking algorithm is the selection of one UAV which assumes the role of the *leader*, a role particularly important for the area coverage task, which is described in the next subsection. The leader is selected by all UAVs using a simple mechanism: since it is assumed that each UAV has a *unique ID*, the leader is selected as the UAV with the *lowest ID* according to a given metric. This choice ensures that, without requiring forms of negotiation or election, all UAVs will select, in autonomy, the same leader.

Covered Area Manager. The main job of the *Covered Area Manager* VAF is the implementation of the area coverage algorithms. Also in this case (since the solution is distributed), each Virtual Drone has its own Covered Area Manager, and its objective depends on the UAV role, i.e. whether it is the leader or a non-leader. In the latter (non-leader) case, this VAF takes only into account the area parts which are gradually covered by the UAV, storing the relevant information in a local *Area Part Database (APD)*. In the former (leader) case, the functionality is a little bit more complex and includes the following steps:

1. the leader polls the APD of all the other UAVs and obtains all the parts already covered by the flock;

2. the obtained data is merged in order to have a global and unique view of the area parts to be still covered;

3. on this basis, the leader plans some possible paths that allow the needed covering and selects the best one (shortest);

4. the leader starts to fly on the chosen path and all the other UAVs, thanks to the flocking algorithm, follow the leader and perform the monitoring functions.

This algorithm is executed periodically, with a period of about $\frac{1}{8}s$; this is required since, if one or more UAVs fail, the area parts covered by them (but not yet transmitted and stored) must be rescanned: thanks to the said algorithm, the path is continuously updated and adapted to changing conditions.

Local Video Processor. We suppose that each twin physical UAV is equipped with a camera sensor that acquires images or video of the monitored area. Such a video is streamed to the *Local Video Processor* that has the task of performing a pre-processing according to the functions specified by the high-level user like, for example, compression, feature extraction, encryption, etc.

Global Video Processor. The *Global Video Processor* receives all the video flows pre-processed by each Local Video Processor and containing the images relating to each monitored sub-area, and compose them to create a global video flow of the monitored area as a whole. The resulting video flow is locally stored by the Global Video Processor for a short time, and then is transferred to the *Historical Video DB* to be stored definitively.

Historical Video DB. The *Historical Video DB* VAF is a simple storage element; it receives from the Global Video Processor, data that not only contains the images themselves but also additional information including metadata, tags and information extracted from the Local Video Processor. All of these data are thus indexed and properly stored in order to be available for future search and analysis.

A further VAF that plays a key role in the whole monitoring system is the ***Mission Manager***, whose job is to control and manage the flight of the UAV flock. For example, it imposes the shape and the speed of the flock, according to the instructions received by the human service operator that is referred to as *Mission Initiator* in Fig. 2.

As specified in [43, 25], the UAV Flight Controllers of all the UAVs need to exchange information. Thanks to the fact that they run inside the same slice, we can create a backend virtual network connecting all the Flock Controllers that, therefore, can communicate with each other by broadcasting their information to be shared with the other UAVs. The same is achieved for information exchange among the Covered Area Managers of all the virtual UAVs.

Interface	Packet	Size	Latency
$I_{PD \leftarrow VD}$	sensor data, lat,lng, altitude, status	60 byte	1 ms
$I_{PD \rightarrow VD}$	motor power	32 byte	1 ms
$I_{Fl \rightarrow Fk}$	lat,lng,roll,pitch,yaw,status	29 byte	$\frac{1}{50} ms$
$I_{Fk \rightarrow Fl}$	V_x,V_y, V_z, ω_z	16 bytes	$\frac{1}{50} ms$
$I_{Fk \rightarrow A}$	Image bounds	16 bytes	$\frac{1}{8} ms$
$I_{A \rightarrow Fk}$	Path bounds	16 bytes	$\frac{1}{8} ms$
$I_{Fk,Bc}$	lat,lng,hgt,yaw,status	17 byte	1/8 s
$I_{Fk,L}$	compressed image from camera	1 MB - 10 MB	best effort
$I_{L,G}$	compressed image from camera	1 MB - 10 MB	best effort

Table 2.1: Interface list and description

Now, considering the flocking formation and maintenance algorithm and the data to be exchanged through each interface shown in Fig. 2.2, we have derived the delay requirements for each interface shown in Table 2.1, together with the kind and the size of exchanged data.

2.3.2 Interface Definition

The interfaces defined between VAFs are listed in Table 2.1 together with communications requirements that have to be negotiated with the *Network Slice Orchestrator*.

$I_{PD,VD}$ is a bi-directional interface between the Physical Drone and the Virtual Drone. In the uplink direction, the Physical Drone sends all the data related to inertial and position sensors, i.e. gyroscope, accelerometer, magnetometer, barometer and GPS (for the first tree sensors, a triple of values is acquired, one for each geometric axis X, Y and Z). In the downlink direction, the Virtual Drone sends the power of the motors computed by the stabilization and navigation algorithm.

The $I_{Fl,Fk}$ interface connects the UAV Flight Controller and the Flock Controller. Here exchanged data are the ones that are relevant to the flocking algorithm: the uplink conveys position data while the downlink carries the set points for translational and angular speeds.

The $I_{Fl,A}$ interface connects the Flock Controller to the Covered Area Manager; here the uplink is used to carry the information about the area portions which are

gradually covered, and the downlink, which is used only by the leader, transmits the path planned to perform area coverage.

Interfaces $I_{Fk,Bc}$ and $I_{A,Bc}$ are two broadcast communication interfaces used to share data among respectively Flock Controllers and Covered Area Managers. They must support the same bandwidth and latency of $I_{Fk,Fl}$ and $I_{Fk,A}$.

The Interface $I_{PD,L}$ is used by the Physical Drone to transmit the captured images to the Local Video Processor, the requirements of such an interface are therefore similar to (but not the same of) those of traditional video channels, even if real-time capabilities are not requested since the video must not be displayed. The same characteristics are required by the $I_{L,G}$ interface, which connects each Local Video Processor to the relevant Global Video Processor.

The $I_{M,Flock}$ interface is used by the Mission Manager to control the flock, by sending to the Virtual Drones mission-specific parameters, such as area bounds, flight altitude, max flight speeds, etc.

The $I_{O,M}$ interface allows the human operator to configure the mission by sending the configuration parameters to the Mission Manager. On the other side, the Mission Manager periodically provides the Human operator with the updates about the mission progress.

Finally, the $I_{M,O}$ interface allows the Mission Manager to negotiate the network slice parameters, in terms of computation, storage and latencies, with the Network Slice Orchestrator, not only at the slice setup, but also at run time if some modification is required during the service lifetime.

Information contained in Table 2.1 will be used by the Network Slice Orchestrator to decide the placement of the VAFs inside the clouds available in the network, and the links to realize the virtual graph of the overall virtual application service. For example, each UAV Flight Controller must be deployed in the Cloudlet closest to the current access point of the corresponding Physical Drone, while the Global Video Processor and the Historical Video DB must be placed in clouds where high computation resources and high storage resources, respectively. are available. Given that no strict latency requirements are specified for these last elements, the most suitable placement candidates for them are the core clouds.

2.4 Conclusions

This chapter proposes a distributed platform for an aerial video-monitoring service realized by leveraging on the Tactile Internet slice of a 5G communication system. The UAVs providing this service are organized in flock, and their control is performed by a chain of virtual application functions (VAF) running on the ground, in clouds at the edge of the network. Very hard control loops, as the one needed to control the engines of each UAV, which are usually realized by control functions installed on board of the UAVs, can now be realized outside the UAVs, i.e. on the ground, only thanks to the ultra-low latency and high reliability peculiarities guaranteed by the Tactile Internet. The resulting deployment of computations in the edge provides many advantages, in terms of scalability and fault-tolerance, and avoids processing latencies, due to communication links, that represent an important drawback in traditional solutions. Dimensioning the maximum number of UAVs that can belong to the same flock, their minimum distance and the maximum speed for a given UAV are considered as a future work. Another future work will consist in the definition of autonomous placement, resource allocation and orchestration policies that will be able to decide where placing VAFs, and the amount of networking, storage and computing resources providing to each of them, even in presence of mobility, that is, immediately after handover events, which can be very frequent during a flock flight.

Chapter 3

A Self-organizing Network Protocol for LoWPAN Networks

3.1 Introduction

In the last few years, we witnessed an impressive growth of the *Internet-of-Things* technology[46, 47], with the consequent widespread of IoT devices that, used in any kind of large-scale applications, are employed to form very big networks and manage very large quantities of data.

Among IoT applications, the theme of *Smart Cities* [48] is becoming increasingly popular: in such a context, IoT devices are used to exchange information about public transportation, traffic, air quality (pollution), etc., and, in the near future, it is expected that also traffic lights will be automated according to the historical data collected over time. Such devices can send these information to our cars, allowing drivers (or even cars themselves) to choose the best route, in terms of efficiency (time navigation) and air quality[48]. Devices can be also installed directly in sensitive parts of city buildings, collecting information through appropriate sensors (like accelerometers) with the objective of making the system able to try to predict stability problems or identify events due to natural forces, such as earthquakes.

Also in the context of *smart industries*, IoT technology is giving a great support: indeed, in production chains, not only a timely monitoring of the operating machines is essential but, above all, the just-in-time prediction of failures is very important [49].

However, even if the IoT technology promises to solve the problem of large-scale monitoring, it poses a series of important issues that, in many cases, represent a

big limitation. In IoT applications, the size of the system, in terms of number of nodes, is very large, ranging from hundreds to thousand of devices, thus posing a serious problem in terms of configuration, initialization and maintenance of the network, operations that cannot surely be done using a manual intervention device-per-device. From this point of view, there is not a common or standardized or even state-of-the-art approach, but each specific application follows its own rules and guidelines.

Another critical issue is battery life: basically IoT devices consume a lot of energy when communicating with the access point (that often acts also as a gateway), therefore power consumption must be optimized and balanced otherwise the switching-off of a devices–if frequent–would cause network structure's changes and the consequent waste of energy, thus reducing the whole network's lifetime.

Battery depletion, as well as other kind of node failures, causes the disappearance of that node from the network, with consequences that, in some cases, could be quite harmful (think, for example, to the case in which the failed node acts as a gateway). While human intervention could surely solve the problem, the ability of self-repairing is surely a desirable feature.

Given the premises above, this chapter presents a LoWPAN (Low power Wireless Personal Area Network) network protocol aimed at facing the cited issues. Basically, it supports an automatic network construction (without human intervention) by creating a layered tree structure featuring one *Lead node*, linked with the wireless gateway, some *Middle nodes*, acting as normal nodes and as links between their child nodes and the rest of the network, forwarding packets; finally the *End nodes* (at the leaves of the tree) with the role of sending and receiving data from them to the rest of the network. Such a network structure, as well as the underlying protocol, not only faces the problem of configuration but is also able to balance communication among nodes in order to ensure a fair power consumption. In addition, the protocol features self-repair capabilities since it is able to identify node failures and perform automatic recovery and re-configuration of the network, if needed.

3.2 Related Works

The advent of low-cost, low-powers smart devices has encouraged the development of novel technologies aimed at supporting low-power communications for low-cost devices.

Contiki[50] is an open source, lightweight operating system designed to support dynamic loading and replacement of IoT programs and services. The Contiki kernel is event-driven and provides optional preemptive multi-threading; it provides feasibility for resource constrained environment, as it allows the developer to keep the base system lightweight and compact. Contiki supports IPv4 and IPv6, as well as several low-power wireless standard, as 6lowpan[51], RPL[52] and CoAP[53]. It provides interesting capabilities as ContikiMAC[54] and sleepy routers [55] to support battery-operated routers.

OpenThread [56, 57] was released by Google as an open-source implementation of the IoT standard named Thread[58]. Google released OpenThread to support networking into its own products known as Google Nest (smart devices for home), and to allow developers to easily develop applications. The focus of OpenThread is portability which is achieved by a platform abstraction layer and a small memory footprint. It supports both system-on-chip (SoC)[59] and network co-processor (NCP) designs[60].

LoRaWAN[61, 62] (Long Range Wide Area Network Protocol for Internet of Things) is a data-link layer with long range, low power, and low bit rate specifically designed for the IoT. The LoRaWAN architecture define a "star of stars" topology, where the physical layer, LoRa, enables the long range link. The protocol has positive effects on the node battery lifetime, the network capacity, QoS and security.

As we discuss in the next section, the protocol described in this thesis relies on the 802.15.4 protocol[63].

3.3 The Proposed Protocol

The proposed protocol is built on top of 802.15.4 protocol[63] which allows IoT devices to connect to each other under a single LoWPAN in a large physical area. In the following of this section, we will describe the various elements of the protocol.

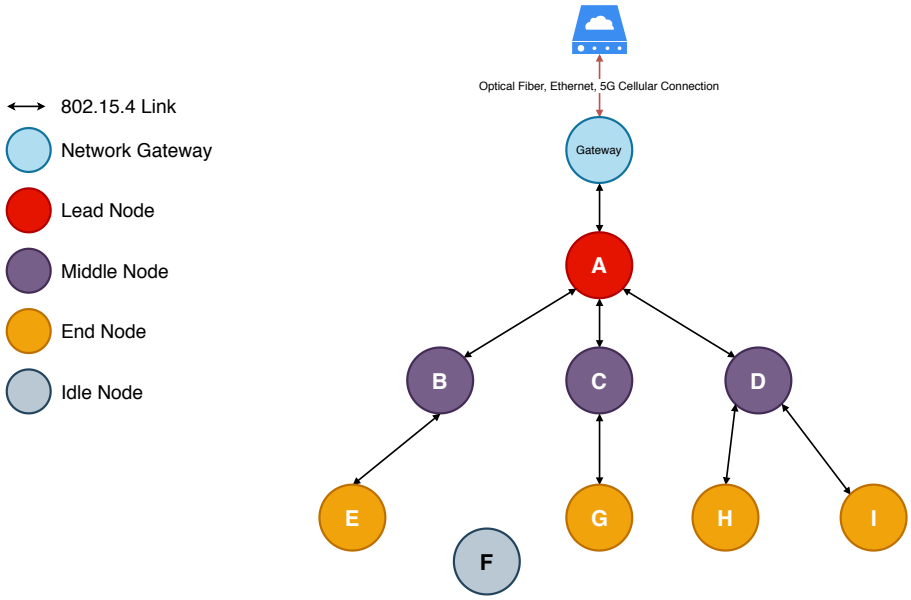


Figure 3.1: A Network Protocol topology and node types.

3.3.1 Network Topology

The protocol is designed to construct the network as a tree of variable depth. In the tree, every node can act as a repeater and can have several down links and a single up link. The protocol provides an upper limit for the down links of each node.

As Figure 3.1 shows, the network is a multi-hop network on which the "lead node" (i.e. the root of the tree) is linked to the gateway. The Lead node is the top node in the network (the root of the tree) and is connected to the Gateway. Only one Lead node can exists in a network and can have only one up link with the Gateway. The Gateway (or hub) is a device with which the IoT devices (so the other nodes) can communicate thought the internet. Typically, this device is connected to the Internet via cable connection (Optical Fiber, Ethernet) or mobile connection (LTE/4G, 5G). The protocol also specifies an upper bound for the number of layers (depth of the tree). The remaining nodes of the tree are labeled End nodes, Middle nodes and Idle nodes. An End node is a leaf of the tree, it will send packets to its own neighbours towards the gateway. A Middle node has a single uplink to its parent and a number of downlinks to its own child nodes. Any node can send packets to any other node through its own neighbours. An Idle node is a node that has to join the network and, therefore, it will attempt to form an up link with a middle

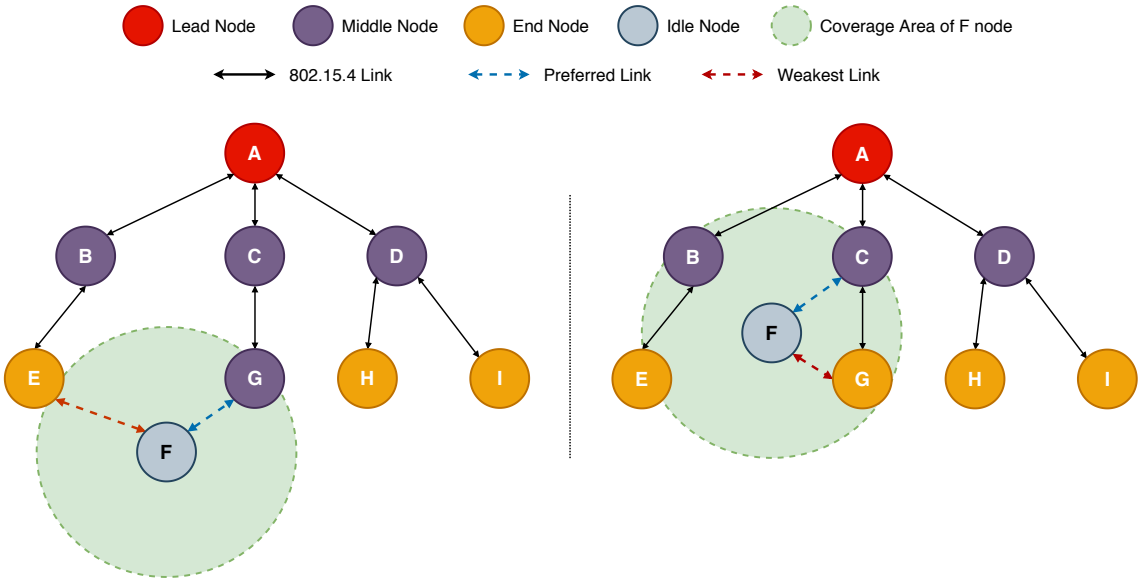


Figure 3.2: Left: the idle node prefer the node with stronger RSSI. Right: the idle node prefer the node with higher layer level.

node.

3.3.2 Presentation Frame

Every node is able to make a downlink transmitting, periodically, a special packet that is a presentation frame. The presentation frame allows the node itself to send its own identity to the other nodes: the protocol uses the device information as well as metadata related to the role of node (lead, middle, end, idle), current layer, maximum number of allowed layers in the network, current number of downlinks (children number) and the maximum number of allowed down links. Other important data are also available such as the uptime of each node and the mean RSSI (Received Signal Strength Indicator); these values are used to quantify the reliability of the node itself. We will discuss in detail this aspect later.

An Idle node listens for these special packets in order to collect a list of nodes which can potentially become its own parent. When an Idle node receives a presentation frame it saves the RSSI. In particular, in order to prevent the creation of weak uplinks, the protocol specifies a threshold for the RSSI: the sender of the presentation frame is added to the list of candidates only in the case the frame RSSI is above the threshold. When an Idle node has more than one candidates, it

will look at the i) layer to which the node belongs to, then the ii) number of down links, iii) the RSSI and iv) finally the node uptime. The Idle node will select the candidate having the lowest layer number (i.e. the candidate that resides in the layer nearest to the Lead node). If there is only one node satisfying this criteria, it will be the new parent, otherwise it has to look at the number of downlink of the several candidates. In this case the node having the lowest number of downlink (i.e. with the lowest number of child nodes) is selected (this choice allows the protocol to balance the communication load of the network). If there are several nodes with the same (lowest) number of child nodes, the Idle node will look at the RSSI value (the highest is preferred) and, finally the value of uptime. An example of this process is depicted in Figure 3.2.

3.3.3 Routing Process

The routing process is very simple. Each node maintains its own routing table containing the MAC (Medium Access Protocol) addresses of all the nodes belonging to the node's sub-network, while the lead node holds the global routing table of the network. As a consequence, any node, having to send a message to another node, can have or not the destination address in its own routing table. In the first case, the node forwards the data packet to the destination MAC address contained in the sub-network owned by the node itself. In the latter case, the node will send the packet to its parent node. The routing mechanism prevents loop back during the middle node selection, excluding nodes that are already present in the selecting middle node's routing table, thus preventing that a node connects to any node within its sub-network.

3.3.4 Lead Node Selection

This process is based on a voting mechanism, involving all the nodes in the network. Each node will broadcast its own RSSI and its own MAC address to each other. In a second phase the nodes will send a broadcast message containing the couple {MAC address, RSSI} with the highest RSSI among those received from all the other nodes, as well as its own couple {RSSI,MAC}. This message is, in fact, a vote for the candidate represented by the first couple {RSSI, MAC} of the message. This

process is repeated until the nodes reach a convergence on the best node in terms of RSSI. There may be the case that two or more nodes have the same (highest) RSSI and the same number of votes to become a lead node; in this case the decision mechanism is mainly based on the candidate's nodes uptime, while the RSSI is no more a discrimination value.

3.3.5 Network Node Connection

In a real environment, devices do not power-on synchronously, therefore the network will be built according to the power-on order of devices. For this reason, in the proposed protocol, the following set of rules holds:

1. When an idle node has to join the network, it must evaluate which role it will cover. In case a Lead node already exists, even if the Idle node has a stronger RSSI with Gateway than the Lead node, it will not start an election to become lead node, to prevent the energy waste during the process; it will join the network minimizing the network's structure changes. Therefore the node will join the network as Idle node and it will connect to the best middle node selected among the received presentation frames, as discussed before.

2. When a node connects to the network (to create an up link to its own parent) it may become a new potential middle node for some other nodes. This is possible with a periodically transmission of presentation frame allowing to check the availability of a highest (in terms of height in the tree) middle node. In this way, the network can automatically rectify itself to guarantee that each link has a high connection quality and the number of layers in the network is minimized.

3.3.6 Node Failure

In the event of a node failure, the protocol holds different strategies, depending on the role of the failing node, which can be A) the Lead Node B) a Middle Node or an C) End Node.

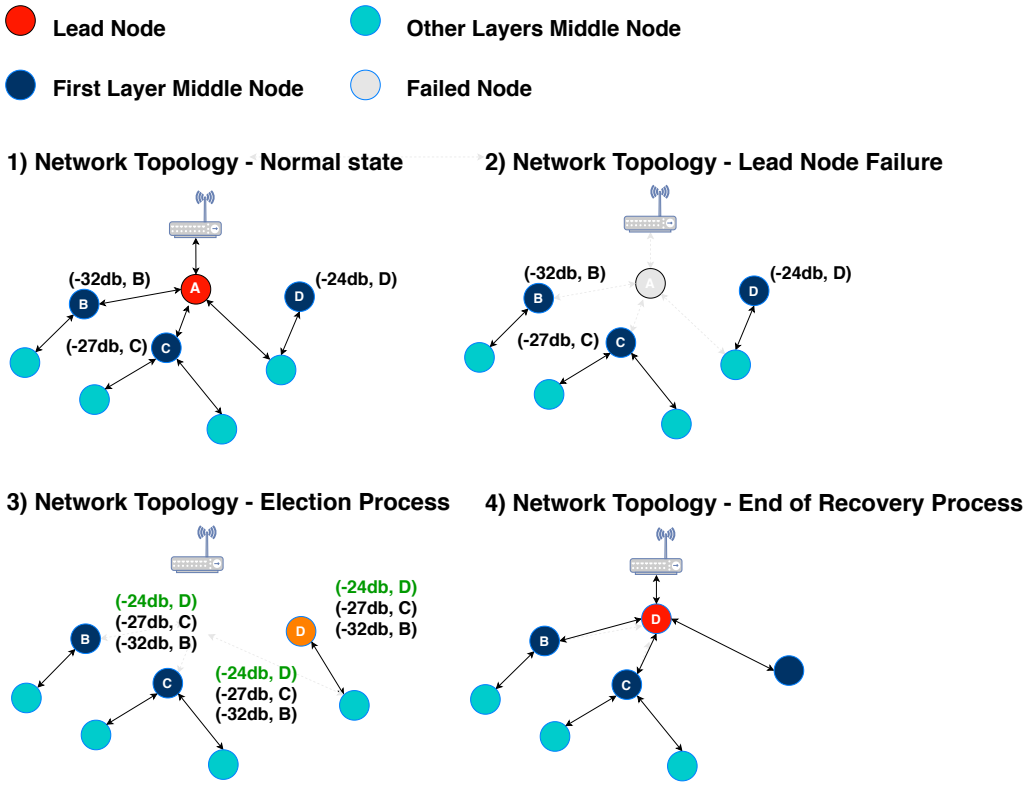


Figure 3.3: Lead Node failure: the recovery process involves only nodes residing in the first layer and thus the only ones with the strongest RSSI to the gateway.

A) When the Lead Node fails (Figure 3.3), the new Lead nodes will be chosen by only the nodes of the second layer of the network. The voting mechanism is the same used at network construction time. Once elected, the new Lead node will connect to the gateway, and the nodes residing in the second layer will link to it. The strategy to limit the election of the Lead node to the nodes residing in the first layer is aimed at minimizing the energy consumption. Moreover, given the construction rules of the network, which are based on the evaluation of node with strong RSSI, it is generally not very useful to involve nodes with an RSSI lower than the first layer's nodes.

B) When a Middle node fails (Figure 3.4), all the nodes previously connected to them become idle for a certain amount of time, waiting that the same node returns available. After a given timeout, it is assumed that the middle node is in failing state, then the idle nodes will look for the first available node to connect to. The choice is based on the mechanism discussed before for the connection of a new node.

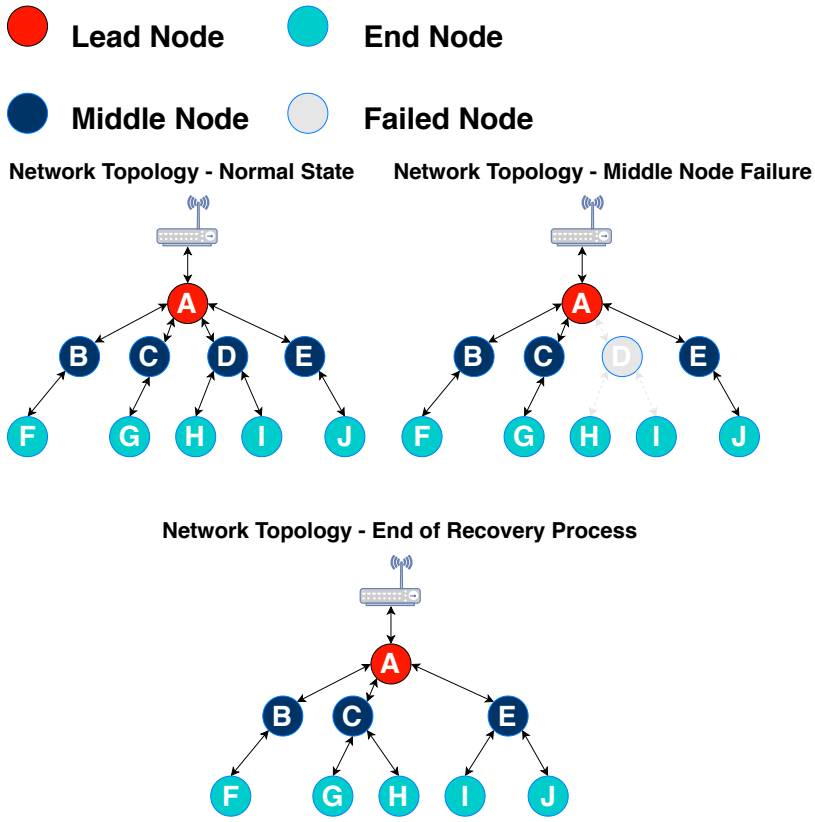


Figure 3.4: Middle Node failure: the Recovery process involves End Nodes directly linked to the failed Middle Node.

It may also arise a scenario where there is no any eligible node (i.e. RSSI under the threshold) to replace the failed Middle node. In this case the orphan nodes will create a link with one of the node residing in their current layer. This will result in the addition of a new layer in the tree. If the maximum number of layers is already reached, then a new layer cannot be added. In this case the system will modify the RSSI threshold in order to allow the orphans to select a new Middle node from the same layer of the failed Middle node.

C) When an End Node fails, the only task to perform is represented by the deletion of the corresponding entry in the routing table of any node previously attached to it.

3.4 Case Study

The approach described in this chapter can be adopted to monitor the production status of hives in beekeeping[64]. Sensors of the LoWPAN network can be placed into every beehive to monitor the internal temperature of the hives and even the sound made by bees, which is an health indicator for them.

Moreover, it is desirable that the LoWPAN network can act as anti theft because beehives are often stolen. To this end, the protocol must be able to adapt and distinguish a fault from a theft; in the last case, a theft can be perceived as an abnormal change in the typical behaviour of one or more node: the gyroscope of the device will record an unusual movement (that excludes a fall from the original position), and start also evaluating the latency of communication. The latency measure is useful to understand how many other hives are being stolen and allows them to form a subnet that lasts as much as possible. Only one node a time is selected to use GPS (Global Positioning System) until it exhaust the battery. All other nodes will follow the same behaviour until all of them will consume their own battery. This represents an optimization to maximize network duration. More in general, every hive is seen as a single node; each node has a precise role inside of the network, and is responsible of its own communication and to forward other node's communication. It has a routing table with a MAC address of the associated node. It is able to auto configure and to adapt and make autonomous decisions in case of fault or theft. Even in this case study, we remark that the human intervention is minimized down to zero, and the communication is strictly related to the basic data exchange, except for the network building process and for the recovery cases. In this way, battery life is optimized to last more without losing important information.

3.5 Conclusions

We described a LoWPAN (Low power Wire-less Personal Area Network) network protocol aimed at facing several issues related to the Internet of Things. In particular we addressed automatic network construction and configuration as well as battery life optimization and fault tolerance. The proposed solution provides an automatic construction of a IoT network which does not need any human intervention. Such

a network structure, as well as the underlying protocol are designed to face the problem of configuration and to balance communication among nodes in order to ensure a fair power consumption. We have described its self-repair capabilities, as well as the process of automatic recovery after a node failure. We also discussed a simple case study in order to show a potential application of the described approach.

Chapter 4

Privacy Preservation enchantment for Delay Tolerant Networks on IoT Environment

4.1 Introduction

Delay Tolerant Networks (DTNs)[65] are characterized by long transmission delays and no assurance to find a complete end-to-end routing path for most of the time. Meetings among nodes are *opportunities* to find new partial paths towards destination. Opportunistic approaches are then adopted to handle routing on this kind of networks: a message can be delivered to the destination node using the physical node movement. The "store, carry and forward" paradigm is used in order to allow communication. To overcome these features, DTNs define a new layer in ISO/OSI stack, between transport and application layers, named *Bundle Layer*. To manage this layer there is the *Bundle Protocol (BP)* [66], that substantially applies the *store and forward* paradigm [67] keeping a packet stored for a time greater than usually, until the packet can be delivered to the next available node toward its final destination.

Many protocols have been designed to implement routing on DTNs. The *epidemic* protocol [68] uses a modified flooding approach to deliver messages. As the name suggests, a message is replicated in every new met node, so every possible route is tested. In term of communication bandwidth and storage memory, this strategy clearly wastes a lot of resources.
Spray and Wait[69] is a modified version of epidemic routing protocol that tries to

reduce overhead bounding message flooding only in the destination area. It works in two phases: in the first one, the so-called *spray* phase, a new created message is distributed in, at least, k copies to specific relay nodes, similarly to the epidemic strategy. After that, if the destination node is not reached yet, the *wait* (second) phase starts: no other copy will be generated and every relay node will deliver the message only to the correct destination.

Others DTNs routing protocols base their decisions on past behaviour: in real life the node movements are not random, but follow a social behaviour. In these cases, there is a non trivial probability that two specific nodes, that met each other in the past, will meet themselves again in the future.

PRoPHET [70] bases its routing protocol on the knowledge of these behaviours. When a node is met, the probability, that it could reach directly or indirectly the final destination is calculated. To implement this approach, every node has to maintain a vector that keeps trace of previous meeting, in order to calculate future meeting probability. This information is shared with other nodes, allowing them to decide whether transfer messages or not.

The exchange of this information among nodes represents an obvious weakness in terms of *Privacy Preserving* [71]. Malicious intruders can collect them to easily rebuild the graph of node contacts and movements, violating their privacy.
Privacy is defined by Clifton *et al.* [72] as "the prevention of personal data usage in a way that negatively impacts someone's life". Preserving privacy can therefore be referred to as a measure of preventing a malicious user to exploit information for identification of a physical being or discover relationships physical being.

The main aim of this research is to present a routing algorithm for DTNs that is able to drive messages from a source node to destination, by using data not related to content or network layouts, with a good approximation and performance. The proposed routing protocol works with blurred data, which are meaningless if considered independently by the context, providing the privacy of the nodes, in the case of leakage by malicious intruders.

4.2 Related Works

Nowadays, *Privacy Preserving* is assuming a role even more important inside digital communications because a lot of data, often related to our daily life, are conveyed into communication infrastructures. For instance, several mobile applications share our personal data for job search or other private purposes, that can be spoofed for malicious intents. Another example could be the context of IoT (Internet of Things)[46]: it is composed by a large number of small devices whose that share a big amount of data which may be also sensitive. In fact, with the increasing of the home-automation, many of these devices can be embedded in our home furnishings. In the context of IoT, DTNs represent a good choice to implement a network infrastructure.

In [73] the authors describe an approach based on multiple paths for each destination to prevent traffic analysis. The proposed architecture uses layered encryption scheme based on a PKI. A message is divided in parts using erasure code and each part follows a different path.

The authors in [74] present two methods based on Bloom filters, that are a space-efficient probabilistic data structures used to represent sets.

Looking at the social-based algorithms can be observed that human relationships are reflected in the contact opportunities of the devices they carry with them. In this case, the routing decisions are based on metrics coming from Social Analysis and characterise the importance of each node on the network. Some well-known algorithms of this category are the BubbleRap [75] and the SimBet [76].

The first one substantially reduces the representation space overhead using a summary vector. The second one uses a Bloom filter in opportunistic networks. When two nodes encounter each other, they exchange the information stored in their buffers to avoid useless transmissions and control redundancy, through a summary vector indicating the packets already received. Unfortunately, Bloom filters introduce some drawbacks like additional processing overhead during each packet routing and the most famous “Probability of false positive”. To reduce this phenomenon, Bloom filters should be used in a very large scale of node sets, which increases the probability that information privacy of some nodes could be violated at the same

time. Another way adopted to keep information privacy is in [77]. It take advantage by an encryption heuristics based on the homomorphic relationship between the data available before and after encryption but for this reason the encounters between nodes have to be direct or the routing paths have to be known (between source, destination and all the relays in the path too) before to send messages. In this case the routing could be hazardous in terms of potentially information shared and and costs.

4.3 Introduction to PRoPHET protocol

To drive messages from source to destination avoiding useless packet replication through the network, PRoPHET [78, 79] applies a probabilistic routing by a specific metric called *Delivery Predictability* (DP). Denoted as $P_i(j) \in [0, 1]$, the DP is established at every node i for each known destination j. The DP provides information about probability to encounter other nodes. When the metric is calculated, a node with a higher value is considered a better candidate for delivering message bundles. For instance, if $P_A(D) > P_B(D)$, then using node A as a carrier towards destination D is statistically better than using node B. The DP is therefore leveraged for the forwarding decisions.
When two PRoPHET nodes have a communication opportunity, initially they enter in a two-parts Information Exchange Phase (IEP). In the first part, the nodes exchange their summary vectors, the *DP tables*. During the second part, bundles are forwarded using the results based on the information exchanged in the first part.

The basic idea of this protocol is to learn how to route bundles analyzing node movements. This strategy is regulated by the following predictability update rule [79]:

$$P_i(j)_{new} = P_i(j)_{old} + (1 - P_i(j)_{old}) \cdot P_{init} \tag{4.1}$$

where $P_{init} \in [0, 1]$.

Consider the example case in which we have three nodes, where node A frequently encounters the node B, which frequently encounters node C. In such a case, A can be considered a good forwarder for messages that must reach the node C, even if the

node A does not encounter C directly. This can be formalized with the *transitive* rule, expressed as:

$$P_A(C)_{new} = P_A(C)_{old} + (1 - P_A(C)_{old}) \cdot P_A(B) \cdot P_B(C) \cdot \beta \tag{4.2}$$

where $\beta \in [0, 1]$ defines the impact of the transitivity rule in (DP). To implement a forgetfulness procedure, PRoPHET provides an ageing rule:

$$P_i(j)_{new} = P_i(j)_{old} \cdot \gamma^k \tag{4.3}$$

where $\gamma \in [0, 1[$ and $k \in \mathbb{N}$.

Privacy lacks

The exchanges of the DP tables provide sensitive data on node behaviours and node mobility in the network. A hypothetical spy node could reconstruct private information, principally by means of the transitive rule (2).

4.4 Innovation from natural context

When somebody learns to speak a language, he is influenced by the country where he is and by the people which live there. Changing place and/or meeting different people, he will also change his way to talk, in terms of tone, accent or syntax. In fact, when we listen someone speaking around, we can guess where he comes from, with some degree of accuracy.
Suppose now to transpose this concept in a network domain, where we have DTNs instead of countries and nodes instead of people. A node can send a message using some peculiar characteristics of destination. These features are specific of the zone and the context where destination node currently is.
Some DTNs use opportunistic routing protocols to exchange messages, but the cost to implement a correct and efficient routing is to share sensitive information on node topology throughout the network. In this paper will be shown an innovative solution

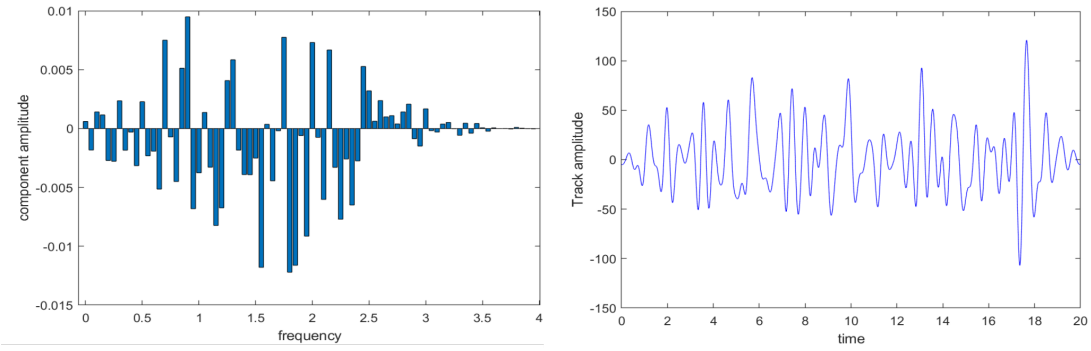


Figure 4.1: Example of a node track, in Frequency Domain (a) and in Time Domain (b) (Note: function b is sampled accordingly to definition)

that minimizes the usage of specific information linked to the nodes behaviours, preferring information related to the environment.

As in human context we speak about vocal tone or timbre [80], in a DTN environment we will use a peculiar track for every node, whose shapes will be moulded by meeting history. The track can be simply thought as a mathematical function. Routing will be based on the similarity of the node tracks. More details will be exposed in the next paragraphs.

4.5 Privacy Preserving Delay Tolerant Network (PPDTN)

The main goal of our protocol is to relax the use of sensitive information from the encountered nodes, using instead more generic, and less sensitive, information. As every person has a peculiar voice timbre, each node needs to have its own shape function that will be called "track". Track shapes aren't permanent. A track will evolve continuously in time, trying to resemble to the ones of the met nodes.

If the tracks of two or more nodes are similar, it means they approximately go around to the same area and they know the same neighbors.

Before to present a mathematical formalisation, we must introduce the main features related to this new proposal:

- *Starting phase*: when a node joins to a network, it needs an initial track. It will be pseudo-randomly generated considering the ones of its neighbors.

- *Evolution function*: when a node meets another, their tracks must be accordingly modified in order to resemble each other. We need a procedure that receives in input the two tracks (the main and the encountered track) and returns the modified track.
- *Matching function*: during the routing phase, a node must decide if the met node can be a good choice to reach the destination, as in any routing protocol based on opportunistic meeting. To implement this, we need a *matching function* that, receiving two tracks as input, returns a similarity value, normalized between 0 and 1.
- *Ageing function*: when a node changes for a long time its location, it must forget the remote influences. Other encounters will help this behaviour, but we need a faster method. The *ageing function* modifies a track toward its initial shape, representing a sort of forgetfulness procedure.

From a mathematical point of view, a track is a continous function with some peculiar characteristics.

$$T(x) : \mathbb{R} \to \mathbb{R} \tag{4.4}$$

To easily maintain and handle it, we introduce the following characteristics:

- the function is defined in [0, n] with $n \in \mathbb{N}$
- $T(x) \in [-m, m]$ with $m \in \mathbb{N}$
- the function must have a strong variability in $[0, n]$

The function becomes:

$$T(x) : [0, n] \to [-m, m] \tag{4.5}$$

All these requirements are introduced in order to correctly implement the proposed solution. The firsts two characteristics allow a simple comparison between two shapes. The last - the high variability in the defined interval $[0, n]$ - is essential to strongly characterize the node and the context.

4.5.1 Starting phase

when a node joins to the network, its track must be randomly generated. The high variability constrain can be obtained leveraging the frequency domain. If we randomly generate some frequency components - in the frequency domain - chosen in conformity to the definition interval, the desired function can be obtained by means the inverse Fourier transform.

Figure 4.1 presents the spectrum (a) in the frequency domain and the correspondent track in the Time domain (b).

Store and handle a function of this kind can be very hard; so we sample the track in the interval of definition, choosing $n+1$ equispaced value for the abscissa:

$$s_i = T(x_i) \qquad x_i = 0..n \tag{4.6}$$

obtaining $n+1$ different points:

$$(0, s_0), (2, s_2), ..., \qquad (n, s_n) \tag{4.7}$$

All the s_i values are then normalized and quantized into the amplitude $[-m, m]$. In the following, we will indicate them with y_i

In this way, a track can be easily stored as a vector of $n+1$ values.

The set with all the points (x_i, y_i) identifies, with a high degree of accuracy, the original function.

4.5.2 Evolution Function

the track exchange during node encounter is a mandatory step to implement evolution and matching functions. To reinforce the Privacy Preserving property it is useful to hide the exact function shape, sending instead a blurred representation obtained from the original one. To implement this, we consider a number k such that:

$$k < n$$
$$n = p * k \qquad \text{with } p \in \mathbb{N}$$

We extrapolate a subset of k values from the original set:

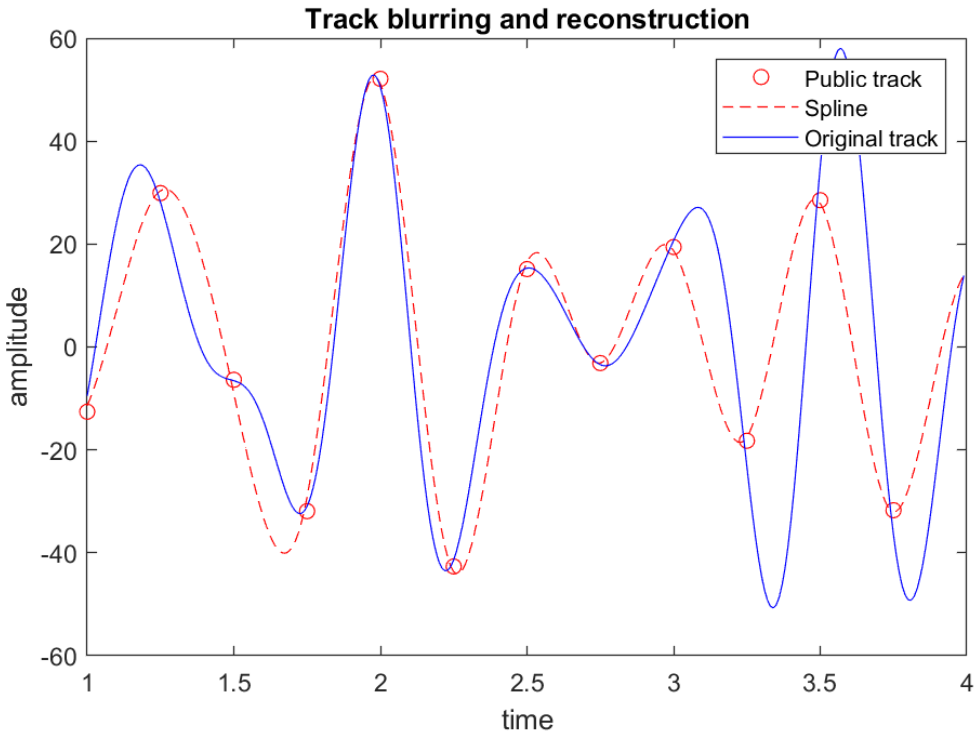


Figure 4.2: A portion of the public track, the relative cubic spline interpolation and the original track

$$x'_0 = x_0 \ \ldots \ x'_j = x_{j*p} \ \ldots \ x'_k = x_{k*p} = x_n \ j \in [0, k]$$

Note that the x'_j values are equispaced (i.e., uniformly taken).

From these values we can derive the correspondent points on the function:

$$(x'_j, T(x'_j)) \qquad j \in [0, k]$$

In order to reconstruct the function, we can use a cubic spline interpolation function [81, 82]; it is a degree-3 polynomial curve that can be derived twice throughout the definition interval, and represents an approximated (blurred) shape of the real track. The blurred representation is what we call the public track of a node.

Figure 4.2 shows, in a zoomed view, the set of points of the public track, the correspondent cubic spline interpolation and the original track. Note that in some parts the approximation well fits the original track, but in other parts some peculiar characteristics are strongly blurred. The precision of the interpolation directly depends by the ratio k/n.

To implement the mutual influence between nodes in case of an encounter, we need an evolution function, that, given a remote public track, modifies the real track of the node.

Let T_A the real track of the node A and $T_{B_{public}}$ the public (blurred) track of B.

We define:

$$f^{evol}(A, B, x) \stackrel{def}{=} \beta \cdot T_A(x) + (1 - \beta) \cdot T_{B_{public}}(x) \tag{4.8}$$

where $\beta \in]0.5, 1[$ is a coefficient that determines the impact of influence of remote tracks on track evolution.

Node B communicates its track with a set of points:

$$T_{B_{public}}(x_j) \quad j \in [0, k] \tag{4.9}$$

Using these points, we derive the cubic spline that interpolates this set. Call it $sp_B(x)$.

Applying the evolution function to node A track, will have:

$$T_A(x_i)_{new} = \beta \cdot T_A(x_i)_{old} + (1 - \beta) \cdot sp_B(x_i) \quad i \in [0...n] \tag{4.10}$$

where $T_A(x)_{new}$ represents the evolved track.

It has been empirically observed that massive execution of the evolution function (4.10) tends to flatten the tracks, minimizing more and more the characteristics. To avoid this behaviour, we apply the normalization procedure, as after the generation phase.

Figure 4.3 shows an example of evolution for a track, applying 30 times the formula (4.10) and the normalization to the track of node A, keeping fixed the other track (node B). The figure is a zoomed part of the entire shape. As we can see, when the two tracks present peaks in the same direction, as at time 1.8, the result accentuates the behaviour. Instead, when they are in opposition, the result tends to a mean value (see time 1.2). Further, the effect of the normalization procedure

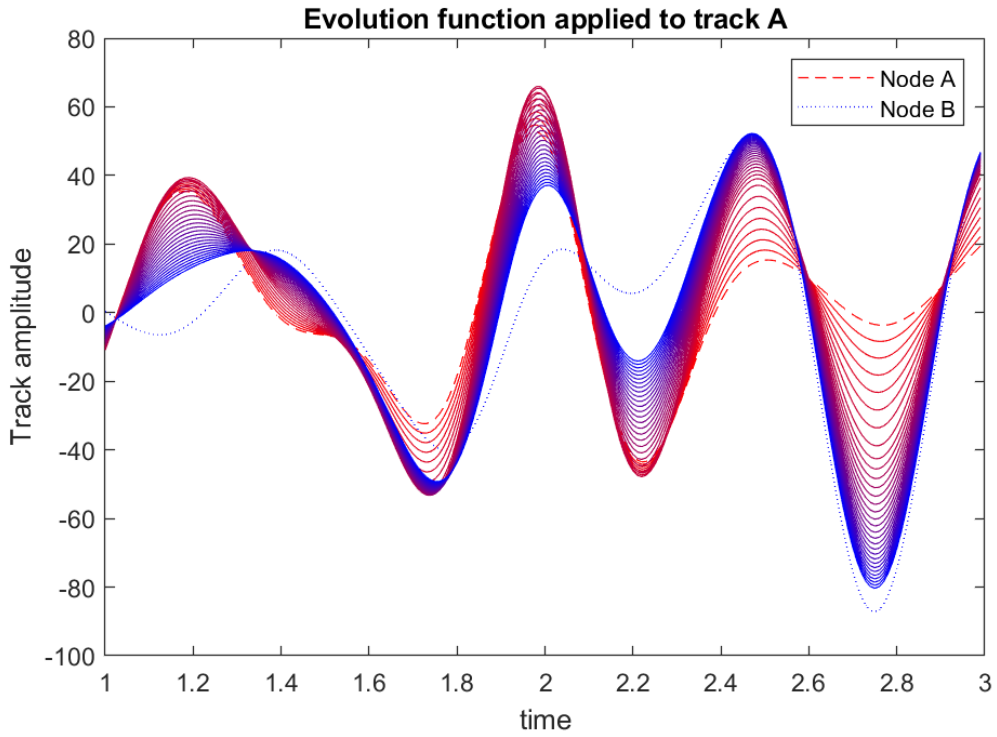


Figure 4.3: Example of repeated applications of the evolution function to the node A track (zoomed view).

is visible in the movements of the peak towards the ones of the track B (see, for example, the interval between 1.0 and 1.6).

4.5.3 Ageing Function

As explained in Section 4.5, we need an ageing function, that represents a sort of forgetfulness method, useful when a node changes its location and its contacts for a long time. This function modifies the track towards the original one. It can be implemented in a similar way of the evolution function. It is be applied with regular time slots.

$$f^{age}(A,x) \stackrel{def}{=} \gamma \cdot T_A(x) + (1-\gamma) \cdot T_{A_{orig}}(x) \tag{4.11}$$

where $T_{A_{orig}}(x)$ is the original track of the node A, as generated in the starting phase, and γ is a parameter defined in $]0, 1[$.

4.5.4 Matching Function

The matching function provides a simple information on the ability of the encountered node to carry data towards the correct destination.

Starting from the profile of the destination and the public track of the met node, the function returns a value in the range $[0, 1]$, where 1 means a perfect match.

Let $T_1(x)$ and $T_2(x)$ the two functions, reconstructed with the cubic spline interpolation from the set of points associated with the destination and the met node. We define it in this way:

$$f^{match}(T_1, T_2) \stackrel{def}{=} 1 - \frac{\sum_{i=0}^{n} |T_1(i) - T_2(i)|}{2mn} \tag{4.12}$$

Every met node that determines a matching function result higher than a predetermined threshold can be considered a good carrier. Moreover, the set of results of the matching function for a given destination, can be used to determine when to stop the research for other carrier, in order to assume *delivered* the message, and delete it.

4.6 Experimental Results

To validate our protocol, we tested it in a scenario of 20 x 20 kilometres with a population of 100 nodes. The range of communication for every node is about 250m. In the scenario a set of zones has been defined. They represent the locations where the nodes move towards and stop for a certain time (600s in average). The dimension of the scenario and the number of nodes have been chosen to avoid that a generic node could meet every other node, but only a limited subset, typically, at most, 15 other nodes. The speed of the movements is variable between 3 and 30 m/s, similarly to human beings.

Every node follows a route that, with some probability, drives it towards a point chosen in a prefixed set. Let $L_i = \{w_1, w_2 ... w_{z_i}\}$ be the list of points chosen for node i (z_i is the size of L_i). When the node reaches the position w_j, we randomly select a value $k < z_i$ and a set of probability $p(x) : \ x \in [1, k]$ such that $\sum p(x) = 1$. Every $p(x)$ value corresponds to a point in the list L_i that follows the point w_j. In other words, the node follows a route that includes all the position of the list in a circular way, but some positions could be skipped in a given round. This behaviour reproduces the real human one.

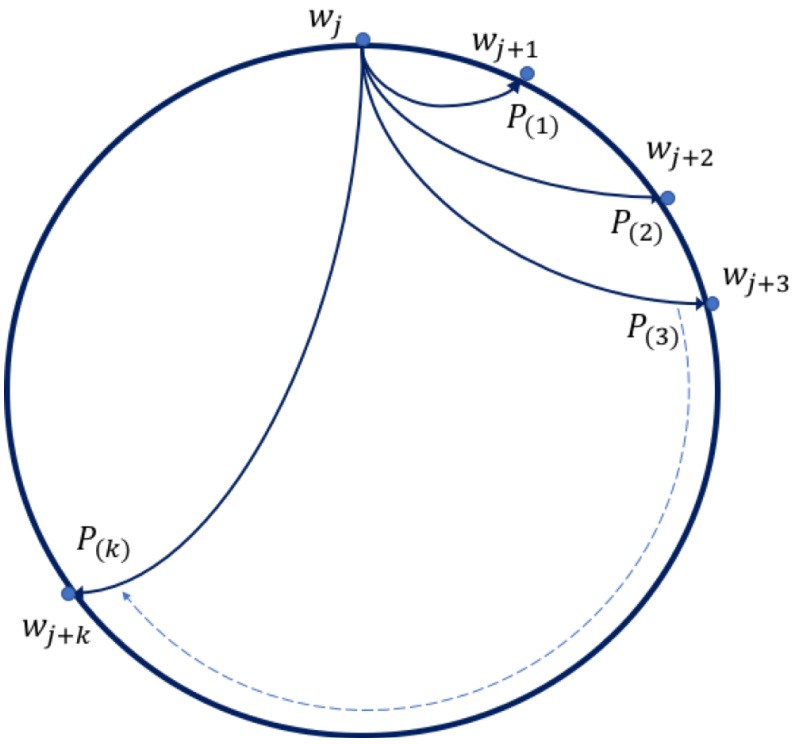


Figure 4.4: Example of movements starting from the position w_j. The value $P(i)$ represents the probability that the next step is the point w_{j+i}

During the simulations, we monitored about 10.000 encounters between all the 100 nodes.

To validate our protocol, we compared its performance with the PRoPHET ones. Figure 4.5 shows the distribution of the encounter probability obtained for all the occurrence of the encounters. For PRoPHET every bar represents the value that the encountered node communicates in order to reach the destination. For PPDTN,

instead, it represents the results of the matching function. We compare these values each other because they are used to choose (or not) the encountered node as carrier. As we can see, both distributions present a very similar shapes. In particular, the number of encounters with low value ($P < 0.70$) is high for both protocols, representing, respectively, 94% and 88% of the total.

To evaluate the actual working phase, we must extrapolate only the portion in the range $0.70 - 1.0$ from this chart, because lower values are discarded by both protocols (i.e. the encountered node is judged as a useless carrier).

In fig. 4.6 the focus is placed in the range $0.70 - 1.0$, considering only the encounters that provide a useful possibility for the deliver, higher then a given threshold. Every point indicates the number of useful encounters with a given probability. For example, we have about 480 encounters for PPDTN and 220 for PRoPHET with probability in the range $0.70, 0.78$. As we can see, PRoPHET shows a limited increment over the value 0.86, forcing to choose a general threshold lower than 0.80, in order to increment the number of nodes classified as "good choice". Instead PPDTN shows a constant increasing trend from 0.70 to 0.94, allowing a more punctual evaluation of the correct threshold. This last consideration can be used to improve the routing when the message is closer to the destination.

Within the simulation we also tested the message delivery performance obtained applying the rules defined for both protocols. After an initial running phase to reach the steady state, the simulator generates a message every 100 seconds on average, with random source and destination. The messages are not duplicated: when a node transfers a massage to another one, the old copy will be deleted. Figure 4.7 diagrams the number of messages delivered for PRoPHET and PPDTN.

As we can see, only a limited number of generated message reaches its destination. This is in accord with the chosen scenario, with a large dimension (400 km^2) and a limited number of node (100). This was done to stress the protocols, highlighting flaws in the routing decisions.

The results show that PPDTN performance tend to double the PRoPHET ones.

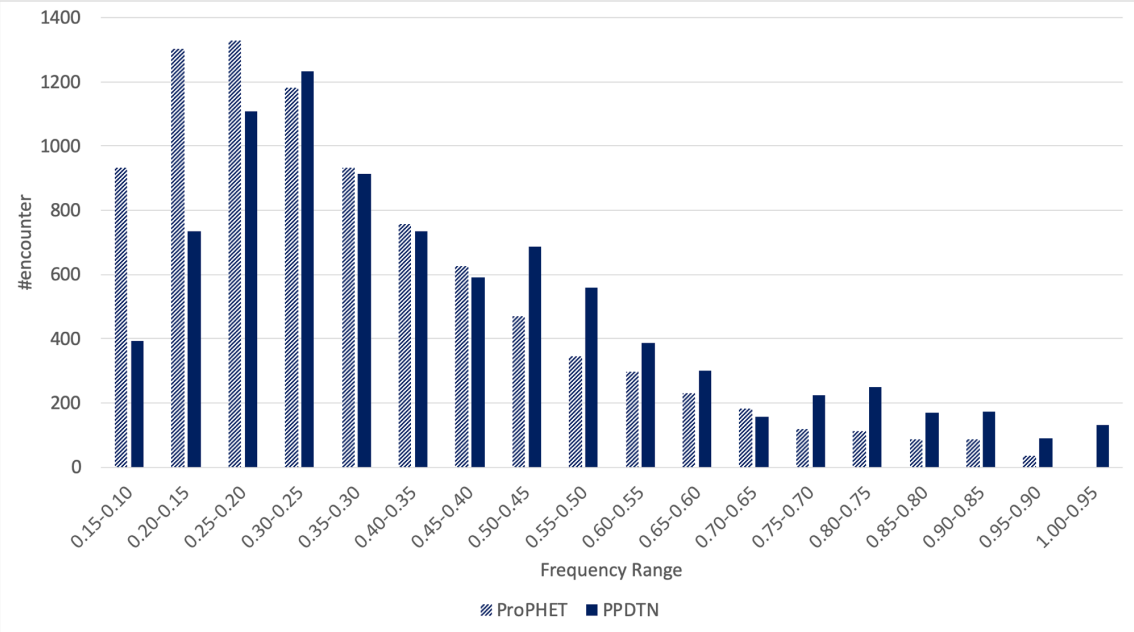


Figure 4.5: Comparison between PRoPHET vs. PPDTN in terms of encounters that can reach a given target.

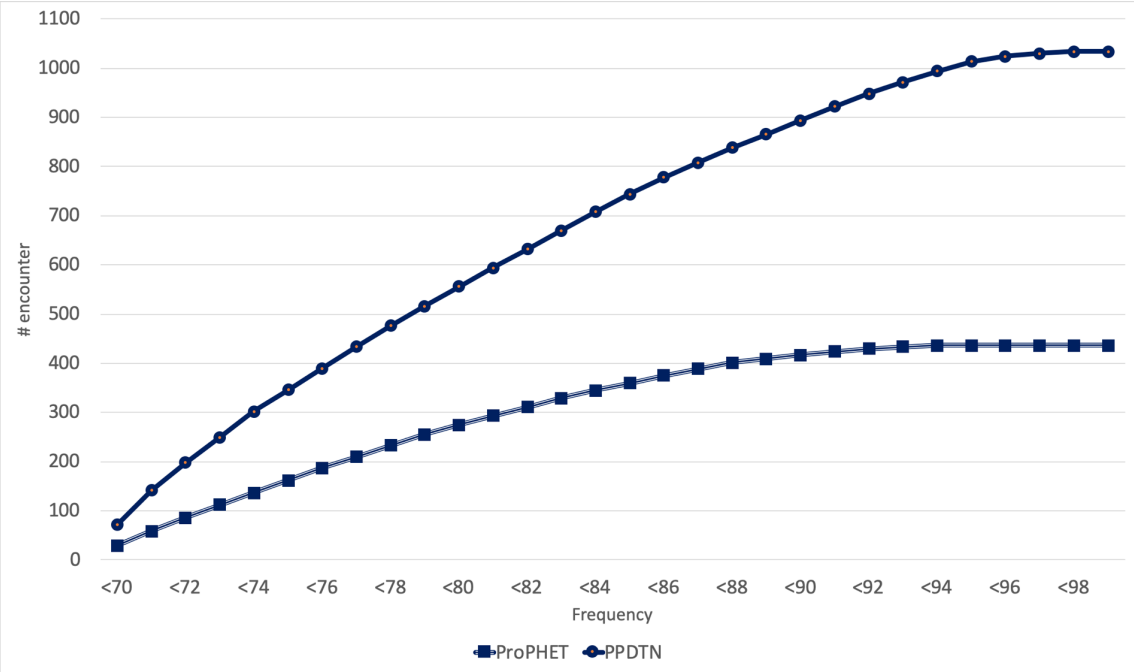


Figure 4.6: Performances comparison between PRoPHET vs. PPDTN in the range of working value (values less than 0.70 have been erased).

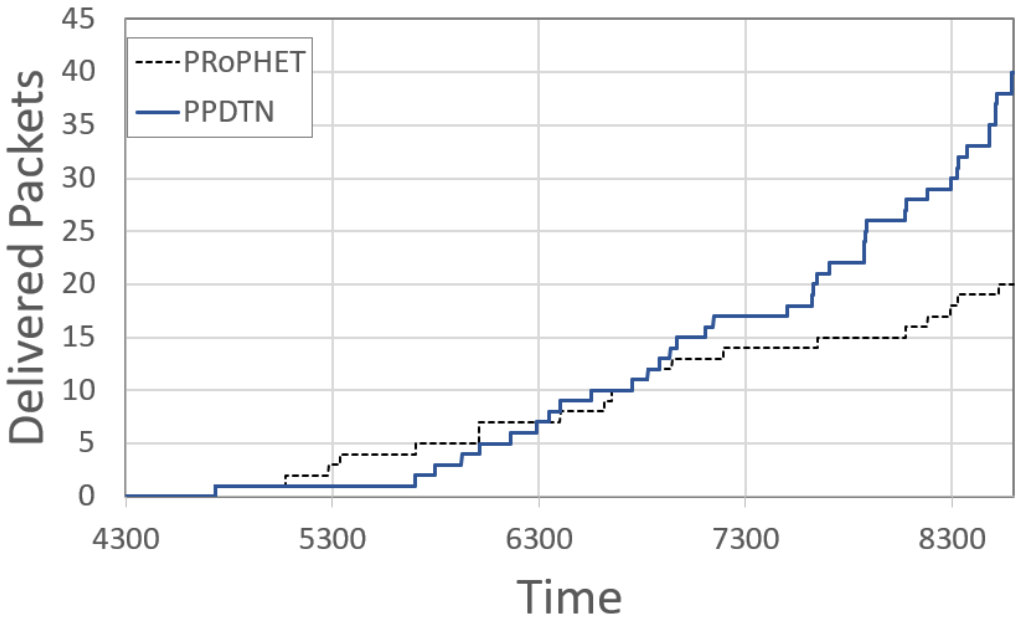


Figure 4.7: Performances comparison between PRoPHET vs. PPDTN in the range of working value.

4.7 Conclusions

DTNs protocols like PRoPHET work to resolve routing problems with forecast approaches but show some vulnerabilities on information privacy, that could become limits for their applicability in case of sensitive data. What was exposed along this paper has to be considered a new way to approach routing protocols in those conditions.

Through this work it has been possible think outside canonical schemes, moving toward a concept related to something that we usually knows and that for this reason we habit to ignore, the features that nature provided us since our existence: graphical shape of the math function as *vocal timber*.

It has been the keystone to realise new solutions introducing new features never used before with a high potential of future developments. To highlight the double effect given by tracks in terms of tool to get the destination and to keep the privacy safe at the same time. This strengths applied to PPDTNs turned out a novel protocol with excellent results in terms of routing performances keeping also our information safe from malicious intentions.

Part II

Simulators and Performance Improvements for IoT

Chapter 5

A Framework for Realistic Simulation of multi-UAV Applications and Networks

5.1 Introduction

The common recent trend of the research in the field of ummanned aerial vehicles (UAVs) is devoted towards *multi-UAV applications* [83, 84]: the challenge is now proposing algorithms and techniques to make *sets of UAVs* achieve a *common mission* in a more or less coordinated way.

A flight mission can vary in objectives and nature, but is basically made of a common *flight plan* that the UAVs of the set must follow; it is, in turn, specified according to the real nature of the mission itself, e.g. as a set of waypoints that UAVs must reach, a certain area of terrain that must be scouted to gather images or other kind of data, etc.

When a set of autonomous UAV is considered, a two-level concept of autonomy has to be taken into account: a first level, the lower one, that we can call the *UAV Level*, relates to stability and flight control and is a matter of the single UAV; a second level, the higher one, that we can call the *Mission Level*, refers to control of the activities of the mission and is the most important part in a multi-UAV application.

Indeed, the *UAV Level* is generally implemented by means of algorithms based on classical state-of-the-art Proportional-Integral-Derivative (PID) control loops, sometimes suitably optimised to improve stability. On the other hand, the *Mission Level*

includes techniques and algorithms that are strongly dependent on the mission type and the mission policies employed, varying from simple forms of task assignment [44, 85, 86], in which a centralised base station instructs each UAV to reach a sequence of waypoints, to more complex forms of coordinated flight, in which UAVs try to self-organise [87, 43, 88, 89, 45, 25] without a central control entity, each one planning its own activities, in order to reach a common goal. In any case, since UAVs must interact with each other and/or with a base station, a communication system is mandatory and, given the nature of the application, it must surely be wireless and based on a technology which is, once again, dependent on the nature of the interactions occurring during the mission. For example, IEEE 802.11 (commonly known as WiFi) could suffice for the centralised case since UAVs must only contact the base station, whereas, when inter-UAV communication is required, ad-hoc networks must be considered. In particular, in the latter case and especially when the set of UAVs is asked to self-organise in a *flock*, the communication channel plays a fundamental role, since it enables the exchange of location or path data among UAVs, letting other UAVs understand where they have to go.

As a result, the development of multi-UAV applications involves considering and integrating concepts and solutions that are proper of control systems, wireless communication, coordination, self-organisation and so on; the natural consequence of this statement is that implementing such kind of systems is quite far from a simple task. However, the most complex issues are *testing* and *debugging*: on one hand, algorithms are designed to run onto physical systems, so the best way to debug them is to perform testing using real UAVs; on the other hand, bugs or wrong tuning of parameters can have catastrophic consequences, such as triggering hazardous behaviour or even induce crashes that, in the best case, only require to repair or rebuild the mechanical frame, but they might even cause injuries to people. To avoid such bad consequences, before running a real system, all algorithms must be preventively verified in order to catch as many bugs as possible (ideally all of them, though full correctness is often impossible to achieve in complex software systems) and fix them, a condition that, however, cannot be reached without proper testing. In other words, we strongly need tests but tests can provoke danger: while this seems to lead to a deadlock condition, a good solution is to be able to *simulate the system* in such a way as to make it possible to assess algorithms, to debug and

to tune parameters.

Dealing with simulations of multi-UAV applications is not a straightforward task. While a large variety of multi-agent simulators exist [90, 91, 92], the key differences lie in the tools' ability to simulate the real system and their accuracy. For instance, NetLogo [90] or Repast [92] simulators are able to support the motion of the entities with a specified speed and direction, but such a motion is performed without taking into account the *physical dynamics* of entities themselves, an aspect that, above all, in the case of UAVs can strongly affect cooperation algorithms. Indeed, UAVs are mechanical systems whose dynamics tend to oscillation or instability, especially in the presence of environmental conditions such as wind or turbulence, hence their real behaviour must always be considered when designing a multi-UAV application.

Communication is another not less important issue in simulation of multiple autonomous entities: typical multi-agent simulators often implement communication in a naïve way (e.g. a simple message-passing), while real systems feature latencies, packet losses and other kind of problems that are typical of real-world networks and, once again, could have a strong impact on the dynamics of the physical system and thus the overall behaviour of the multi-UAV application. For instance, if control algorithms depend on input from other UAVs, high network latency and/or congestion might keep the system from stabilising.

As a natural consequence, the ideal framework for testing a multi-UAV application should be *(i)* to refine your algorithm using the simulator and *(ii)* let it run *directly on the physical system without any modification*. To achieve this goal, the basic requirement is to have a simulation environment that is as realistic as possible in terms of both the physical system and the software stack/APIs. Many tools are currently available: for example, *Gazebo* [93] or *V-REP* are simulators able to simulate the physics of robots; similarly, a variety of network simulators exist [94, 95], with the ability of simulating, in a more or less precise way, wired or wireless networks, along with classical hardware equipment, such as network cards, switches, routers, etc. However, all of these simulators are not designed to work together in an all-in-one integrated simulation environment, which is what we need for a multi-UAV application. And even if some hacks or interfaces could be written to let these tools talk to each other, the main issue is that a *common notion of time* is a mandatory requirement, otherwise the needed realism cannot be achieved.

Given these premises, the objective of this chapter is to present an integrated solution to let a developer simulate, in a realistic way, a multi-UAV application. In particular, the chapter focuses on the usage of some state-of-the-art simulators and builds around them a "glue" consisting in a protocol, a software module called GzUavChannel and some plug-ins for the simulators; this glue aims at integrating these tool and letting them proceed with a common notion of the time. The tools employed are *Gazebo* [93], *ArduPilot* and *ns-3* [94]. Gazebo is a widely used robotic simulator able to simulate the physics of mechanical systems; it also offers a 3D view to observe how the system is progressing in real time. ArduPilot is a multi-vehicle control platform that implements control algorithms (i.e. the UAV Level) for UAV stabilization and flight control; it is used in many UAV-based projects. Ns-3 is one of the most widely used network simulators.

As the work will describe, we integrated such tools by means of a multi-purpose software integrator middleware, the GzUavChannel, that is able to interact with the tools in order to govern their execution and the overall simulation. The way in which the architecture of the integrator is designed provides a high degree of flexibility: while a 3D visualisation environment is provided by Gazebo, it can also be disabled in order to perform batch simulations and gather specific numerical results. It is also fairly easy to switch network models, thanks to the generic nature of the ns-3 plug-in that we developed. Moreover, this chapter will also describe the protocol that we designed and implemented in GzUavChannel to coordinate the various parts of the simulation. As will be detailed, the protocol is designed in such a way as to make it possible to run some components on distinct machines, effectively spreading the computational workload when, in the presence of a high number of UAVs, the CPU cost of simulation becomes too high.

5.2 Related Work

Simulations can be performed with a wide range of tools, some of which are approximate and non-realistic, whereas others are very accurate, in the sense that a simulated UAV runs the same program that a real one would, receiving the same type of input and producing the same type of output.

The simulation tools that are farthest from our approach are the non-realistic ones such as NetLogo [90, 96], which offers a simple point-shaped agent abstraction that can be programmed in a Logo-based programming language, through which agents can move and interact with other agents that are part of the simulation (e.g. run remote commands, read remote variables). Unlike our approach, NetLogo does not simulate neither physics (i.e. agents can move immediately without any constraint or inertia) nor communication channel characteristics such as latency, capacity and so on. Such non-realistic simulation tools are often used during the initial phase of algorithm design, but quickly become inadequate when one wants to start reasoning about a real-world application, and the need for more realism arises.

Some researchers choose to write custom/ad-hoc engines, modelling only the specific aspects they are interested in. Closer to our framework's goal, this type of simulators is often focused on tuning and capturing issues arising from the constraints imposed by the more realistic models. An example of this approach is [97], which simulates physics of fixed-wing UAVs with wireless communication capabilities using custom models. However, custom tools must not necessarily be based entirely on ad-hoc simulation engines, but they can also reuse one or more pre-existing components. For example, in [98] the authors used a custom MATLAB-based physical model along with the OMNeT++ [95] network simulator.

When more than one simulation engine is involved (so-called "co-simulation" scenario), the issue of bi-directionality arises. Indeed, some works simply chain different simulators in one direction only: for instance, in [99] XPlane [1] simulates aerial vehicles and produce flight paths, while OMNeT++ [95] simulates networking according to those paths. In this architecture, although flight paths are processed in real time, the output of the network simulator does not affect XPlane; in other words, the simulation is not bi-directional, because the physical/behavioural part cannot be influenced by the simulated network, as instead happens in our solution.

Other co-simulation approaches, on the other hand, keep all components synchronised and fully aware of each other. For example, [100] proposes a framework for robotics simulations with wireless modelling based on ns-2 or ns-3 [94], using a bi-directional synchronisation method similar to the one we are proposing. For a more complete taxonomy and a survey on co-simulation, [101] is a useful resource.

[1] http://www.x-plane.com

The Gazebo robotics simulator [93] has been used in several UAV-related works, such as [102], often coupled with ArduCopter or the PX4 flight stack [103]. The DroneKit library is also often used, either alone (using its built-in "software-in-the-loop" engine) [85] or in conjunction with ArduCopter. However, in these projects, the issue of communication (and thus the simulation of a network) is not addressed.

As reported above, there are a relatively large amount of works on robot simulation. Nevertheless, a small amount of these robotic simulators allow the simulation of the network used by virtual robots.

In [104] a tool is presented that allows the simulation of multiple types of robotic systems interacting through a communication infrastructure. It integrates the ARGoS robotic simulator and ns-3, providing a scheduling system with the task of synchronizing the robotic simulation, the radio communication and thus the transfer of any data packets to/from a robot. This tool works similarly to our framework. However, the core of ARGoS and ns-3 is strongly modified so the impact of the integration task is really high, whereas our solution is much less invasive. Moreover, the behaviour of the robots must be written using an ad-hoc C++ API, thus the porting of the software onto physical platform would require a rewriting/refactoring process. Conversely, our approach lets a developer port her/his software to a real UAV in a seamless way. Finally, this framework does not present any form of distributed simulation.

PiccSIM [105] is a simulation platform for *networked control systems (NCS)*, i.e. control systems interacting through a network (which, in the case of PiccSIM, is a wireless one). It uses Matlab/Simulink to simulate the control systems and ns-2 to simulate the network. The simulator platform provides a graphical interface tool with which the developer can configure the entire Simulink and ns-2 simulation environment. The simulation platform is intended to run onto a single PC but it can also be split in a maximum of two nodes, while our solution is able to distribute the workload in several nodes.

Player/Stage is a software tool used to simulate multi/distributed robotic systems; it is the precursor of Gazebo. The software tool is composed by two components: *Player* and *Stage*. The Player is a server that expose the sensors of the robot over the network, providing a unique interface for client applications. The Stage part is the robotic simulator that can simulate a large population of robotics

systems, sensors and environmental object. The main objective of Stage is to enable a rapid development of control software that can be used in real robots, as well as to enable experiments without any real hardware or environment.

Another integration middleware worth citing, in the context of robotic applications, is ROS (Robot Operating System). In spite of what the name suggests, it is not a real "operating system" (like e.g. Windows or Unix/Linux), but a communication middleware that lets robotic components (i.e. sensors, actuators, control programs, etc.) interact and exchange data using a common API and a common message format. Data exchange is based on a publisher/subscriber model, that is particularly suited for a robotic scenario. ROS is widely used in robotic research and also in simulation (Gazebo can interact with ROS), even if the use of common message formats adds an unavoidable overhead due to data serialisation, that undoubtedly affects the performance of the applications. ROS is a quite interesting approach to provide integration among components; however, there are no ROS packages supporting interaction with a network simulator.

Finally, a survey on UAV applications, from a networking perspective, can be found in [106].

5.3 System Model

In this Section, an overview of the model of the system used in a multi-UAV application is presented, with the objective of letting the reader understand the various components of a multi-UAV scenario and their role in the context of the application. Such components are depicted in Figure 5.1 and described below.

The basic component is obviously the UAV. As for the airframe, while any type of configuration is possible, researchers tend to employ *multi-rotors* (such as quadrotors), since they are more flexible than fixed-wing aerial vehicles; indeed, they feature a high degree of maneuverability since they support vertical take-off and landing, and translated flight in all directions.

In order to let a multi-rotor fly properly, a *Flight Control Unit* (*FCU*) is always employed; it is an electronic board equipped with a microcontroller, the proper drivers for the motors and a set of inertial and geographical sensors[2].

[2]Accelerometers, gyroscopes, magnetometers, barometers and GPS.

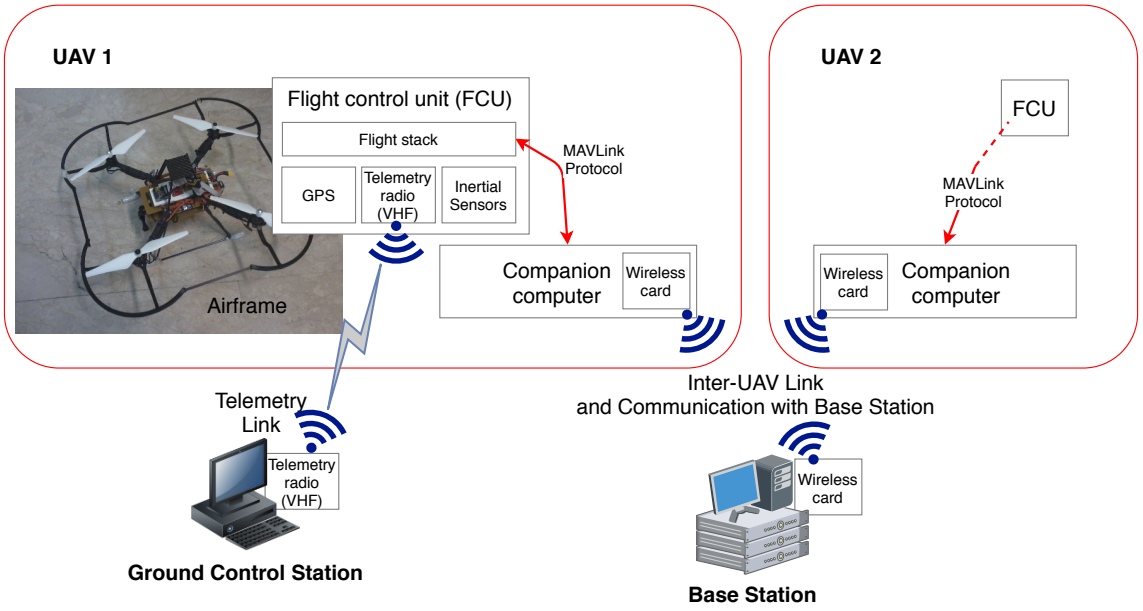


Figure 5.1: Architecture of a UAV and other components of a multi-UAV application

The FCU runs a *flight stack*, which implements the *UAV Level* autonomy cited in Section 5.1. In this sense, there are many ready-to-run products available on the market, as well as several open-source solutions; as an example, one of the most used hardware for UAVs is the PixHawk control board, which is usually equipped with the PX4 [103] or Ardupilot flight stack. In any case, the developer is not asked to consider low-level control issues since such solutions already do the job for her/him.

FCUs often provide some communication interfaces for telemetry, flight emergency control and FCU setup/monitoring. In some cases, a VHF radio modem is included, to interact with a *Ground Control Station*, i.e. a regular PC running a monitoring GUI that allows to perform calibration tasks, read and modify FCU parameters, check the status of the UAV batteries, etc. The *Ground Control Station* only exists for safety purposes and does not contribute to the execution of the mission. The FCU also offers a communication channel that can be used to connect an on-board *companion computer*, i.e. an embedded platform such as a Raspberry PI, Odroid, etc., that implements the high-level logic of the mission, i.e. what we called the *Mission Level* in Section 5.1. This is the hardware platform in which the multi-UAV application runs (for the part related to the single UAV) and, for this reason, it must be equipped with a wireless communication interface to perform

data exchange with other UAVs and/or a possible *Base Station*.

This last component, i.e. the *Base Station*, is an off-board (ground) system (made of one or more computers unrelated to the Ground Control Station) that has the role of managing and controlling the whole mission. This is performed by being in continuous contact with UAVs, and its characteristics depend on the specific multi-UAV application. Depending on the mission algorithm, there might be no *Base Station* at all, as is the case for self-organising UAV flocks.

5.4 Simulation Tools

All the elements described in Section 5.3 must have a relevant corresponding block in the simulated scenario. We propose a solution based on the integration of three existing tools, each implementing a specific function. The physics of the mechanical parts (i.e. airframe, motors and propellers) is simulated by means of *Gazebo*; *ns-3* is used to simulate the network; as for the flight stack, we employed *ArduPilot*.

5.4.1 Gazebo

The Gazebo[3] robotics simulator [93] is designed to be as realistic as possible in the modelling of the simulated environment and in the response of the objects and the sensors contained in it. It has been used in several UAV projects, such as [107] and [102].

Gazebo's software architecture is modular and can be extended by *plug-ins*. According to what aspect of the simulation is to be controlled, Gazebo defines several types of plug-ins. A Gazebo plug-in consists in Linux shared library (`.so`) which is loaded through an XML file. Plug-ins can not only interact with the simulation (using Gazebo's APIs), but also interact with external processes using the underlying operating system's mechanisms (e.g. IPC, sockets and so on). As will be described later, we heavily used this possibility in order to synchronise Gazebo's world with a number of external processes.

Gazebo maintains a clear separation between the core of the simulation (the *gzserver* program) and the 3D visualisation engine (the *gzclient* program). Indeed,

[3] http://gazebosim.org/

it is possible to only launch *gzserver* and run the simulation headless (i.e. without any visualisation, which is useful to collect statistics during simulation campaigns) or, conversely, several *gzclient* instances can be run in order to observe the simulated environment from different points of view.

Being a well-known and *de facto* standard simulation environment, there is a wide library of both built-in and third-party readily-available Gazebo models and plug-ins. For instance, the UAV model that we use for our simulations is the built-in *iris_with_standoffs* model, our code that runs within Gazebo (the GzUavPlugin, see Section 5.5) reuses part of the built-in *ArduCopterPlugin*, and the thrust of the propellers is modelled using the built-in *LiftDragPlugin*.

5.4.2 ArduPilot and DroneKit

The ArduPilot project, originally developed by an hobbyist who later co-founded one of the biggest drone companies, consists in a variety of subprojects, providing control algorithms for autonomous vehicles such as heli/multicopters, fixed-wing planes, submarines and ground vehicles. The module that implements control loops for multicopters is called ArduCopter, and it supports several types of frames (e.g. quad and hexa). A similar project, which ArduPilot shares some code with, is PX4 [103].

Both ArduPilot and PX4, which are intended to run on a FCU, support the connection of a *companion computer*, that is an external on-board system (usually a Linux-based system-on-a-chip) that can retrieve the status of the vehicle in real-time (pose, battery voltage, RC inputs, ...) and send position and velocity setpoints. While low-level control algorithms (e.g. flight stabilisation, real-time pose estimation, simple GPS path-following) are executed by ArduPilot (or PX4) within its dedicated microcontroller high-rate inertial sensors, *companion computers* are usually employed to implement higher-level algorithms that require more complex lower-rate sensors. For example, a target-tracking algorithm would usually be implemented by connecting a camera to the *companion computer*, which runs the computer vision algorithms and then continuously sends velocity setpoints to the ArduPilot microcontroller.

The communication between the FCU and the companion computer happens using a standardised protocol called *MAVLink*[4], usually over a serial channel. MAVLink is a complex protocol, which forces the *companion computer* programmer to deal with a lot of low-level implementation details. In order to hide this complexity, high-level wrapper libraries were created to simplify the task of programming autonomous vehicles. One such library is *DroneKit*[5], which is especially targeted at controlling ArduPilot-based FCUs from Python programs. It automatically configures the serial port and exposes data received via MAVLink through global variables, while also offering an easy-to-use API to send position setpoints. The MAVLink protocol can also be used to remotely control the vehicle from a Ground Control Station by means of wireless "telemetry radios", that are usually employed to monitor the overall status (e.g. receive alarms in case of system failure) and send emergency commands.

Lastly, another interesting ArduCopter feature is the support for SITL (Software-In-The-Loop) simulations, i.e. a configuration in which the target platform is not a real object, but a software simulation of it. When run in SITL mode, ArduCopter offers that same MAVLink control channels a real UAV would offer (albeit via TCP socket instead of a serial port), enabling the validation of high-level algorithms in a simulator using the same MAVLink messages that would be used in the real UAV.

5.4.3 Network Simulator 3

Network Simulator 3 (ns-3) [94] is a simulator capable of simulating several types of network infrastructures and network protocols. It is the evolution of ns-2, one of the most popular network simulators that has been widely used for research and education on the Internet and other network systems.

Its core is written in C++ (with optional Python bindings), and its vast library of built-in modules offers models for technologies and protocols such as CSMA links (Ethernet), bridge communication, WiFi (802.11 links with beacons and ad-hoc modes), network mesh (802.11s and "Flame") and low-energy wireless communication (802.15.4 LoWPAN).

[4] http://mavlink.org
[5] http://python.dronekit.io

Simulations are programmed by means of so-called "simulation scripts", C++ programs that create instances of nodes and assign each of them a network stack and an optional mobility model (describing how a node moves in time). Such scripts would ordinarily also define how node behave up to (simulated) application level; however, in our case, the application logic belongs to the High-level logic processes, and script-defined stacks have to stop at transport level. Similarly, our framework takes over UAV nodes' mobility models to keep their position synchronised with Gazebo.

5.5 The Integrated Simulation Environment

The tools presented in the previous section constitute the basic blocks to build a complete simulation environment for a multi-UAV application; however, they are not designed to work together in an integrated manner. In this sense, there are two main problems that must be faced.

The first problem regards the way in which such tools can exchange data. Since all of them have different interfaces, a proper "translator" must be written, each using the relevant API of the specific tools, to expose data in a common format. This is not particularly challenging and it is a matter of good code writing.

The second, but most important, aspect is tied to *clock synchronisation*. All the tools employed are designed to provide the most realistic simulation as possible; however, each of them includes its own notion of *time*, that is rather different than wall-clock time and mostly tied to the events they simulate. What we need, instead, is the integrated simulation environment to have a *common notion of time*, otherwise the requirement of realistic simulation cannot be met. For this reason, the said "translators" must interact with each other and with the various tools in order to make them proceed, during the simulation, in a *strictly synchronised way*.

5.5.1 Basic Components

The software architecture of the integrated simulator is composed of several modules, each referring to a specific component of a UAV in the real scenario. Each UAV is represented by the following software modules which are shown in Figure 5.2

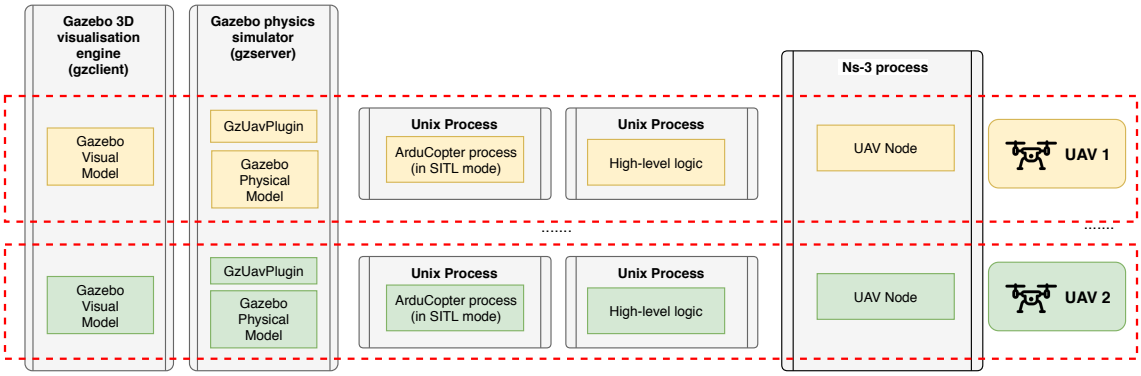


Figure 5.2: Software Components and Architecture of the Integrated Simulator

and described below: *Gazebo Visual Model*, *Gazebo Physical Model*, *GzUavPlugin*, *ArduCopter process*, *UAV node* and *High-level Logic* (or *Mission-level Logic*).

The *Gazebo Visual Model* and *Gazebo Physical Model* represent the definition of the frame and inertia of the UAV, which, in our experiments, is a quad-rotor VTOL aerial vehicle. Both models are coded in the same XML definition file that Gazebo uses to display the UAV in the 3D scenario and to simulate its physics. In our framework, we used the definition file of a quad-rotor, called *iris*, that is available in Gazebo's own model library.

In order to let the simulated quadrotor fly, the frame must be "connected" to the ArduCopter flight stack; to this aim, ArduCopter must obtain proper information on *frame pose* (geographical coordinates, Euler angles and Euler angle rates and must be able to drive propellers by sending proper commands to motors. The *GzUavPlugin* component is a Gazebo plugin that has this specific objective. Derived from the ArduCopterPlugin which is available in the Gazebo package, we suitably modified it in order to include synchronisation of the activities with the other parts of the integrated simulation environment.

The *ArduCopter process* is a stand-alone process running an instance of the ArduPilot flight stack, specifically compiled to run on a PC-based platform (and not a real FCU) and to support the "software-in-the-loop" (SITL) mode. By interacting with the *GzUavPlugin* it is able to control the stability and the flight of the simulated UAV.

The *UAV node* represents the communication end-point, that is the *wireless interface* of a companion computer placed in the UAV. It is in practice an instance

of a C++ class that runs within the ns-3 process and enables the simulation of the wireless communication channel. In our implementation, we support both IEEE 802.11 and IEEE 802.15.4 wireless standards, and the type of communication can be selected when the simulation is launched.

The last part is *High-Level Logic (HLL)*, which is the module running the user-defined software that implements the multi-UAV application. In our case, it is a Python script that exploits services provided by two Python APIs: `DroneKit`, the library already cited in Section 5.4.2, and `ns3interface`, a library written by the authors to let the Python program interact with ns-3. A Python script contains the HLL of the behaviour of a *single* UAV, so, for each UAV of the application, a different Python process must be run; this implies that, when the application is ready to be ported onto a real scenario, the source code of the behaviour can easily be run on the companion computer of each UAV without (or with very few) modifications.

As Figure 5.2 clearly shows, a specific instance of each module is created for each simulated UAV, but all these instances, while constituting altogether a single UAV, do not run in the same environment and each one is executed inside its specific simulation tool: the *Gazebo model* and the *GzUavPlugin* run inside the Gazebo process, the *UavNode* runs inside ns-3, while *ArduCopter* and *High-Level Logic* are executed as stand-alone processes.

5.5.2 The GzUavChannel

In order to support the synchronization requirements dealt with in the previous section, the integrated simulation environment we propose includes a basic component called GzUavChannel, which is depicted in Figure 5.3 together with the links and connections with the other components of the framework[6]. It is a *middleware component* that has multiple aims.

Its first objective is **clock synchronization**. The GzUavChannel has the responsibility of capturing the time tick generated by Gazebo, which is our main clock source, and distributing it to all the other components, in order to allow the evolution of the simulation with a common knowledge of time.

[6]Figure 5.3 shows an environment where two UAVs are simulated; if more entities have to be considered, it suffices to replicate the various modules accordingly.

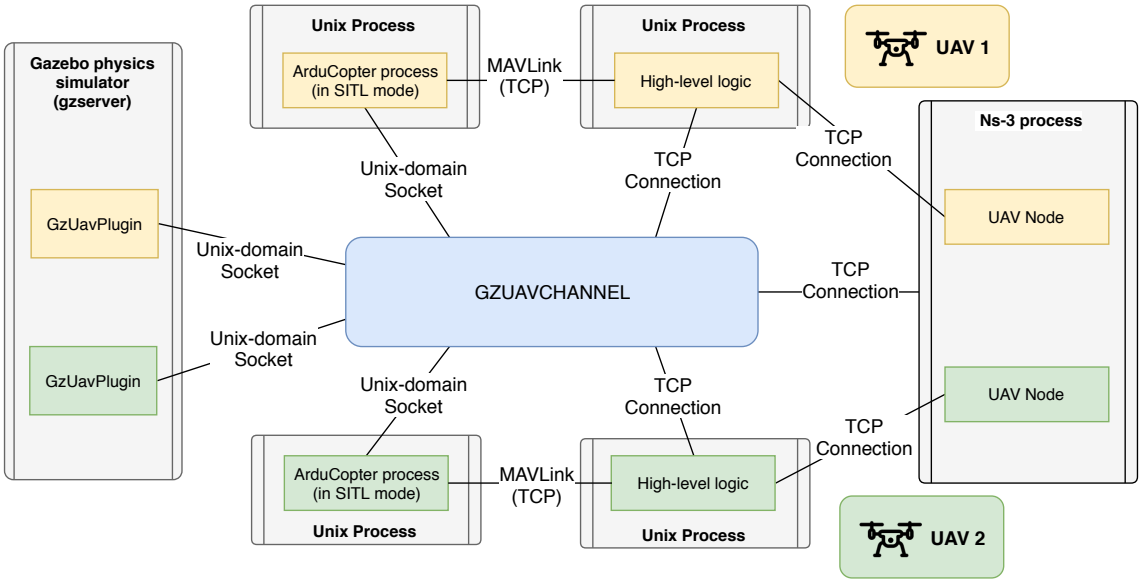


Figure 5.3: Relationships among Components and the GzUavChannel

The second objective is **data exchange**. It acts as a data-bridge among GzUavPlugins and ArduCopter processes; this bridging is the key to handle time synchronisation, but it is also required to support the distribution of the simulation in several network nodes.

Distribution Handling is indeed the third objective of GzUavChannel. As it will be discussed in Section 5.5.4, when the simulation is split over different network nodes (in order to spread the workload), each node hosts an instance of GzUavChannel and all of them cooperate to synchronise the overall activities.

As Figure 5.3 shows, the components of the framework not only interact with GzUavChannel but also directly with each other. In particular, communication between ArduCopter and High-level Logic instances is directly performed through a TCP link that encapsulates the MAVLink protocol (via DroneKit), while another TCP connection is exploited to carry data packets that must be sent over the network simulated by ns-3. In this case, the use of TCP links is required to allow the splitting of the various components over different machines of a distributed environment.

5.5.3 Timing and Synchronization

A basic aspect of the simulators of dynamic systems is the way in which time is handled; this depends of the specific type of system that has to be simulated and, as a consequence, of the laws governing it.

Physical systems, like those simulated by Gazebo, are modelled by means of differential equations that govern their dynamics and kinematics; in order to be simulated, those equations must be discretised and then implemented in software. Discretisation is performed according to a certain *sampling time* that constitutes the time granularity of the simulation: for each iteration, we assume that a time interval equal to the sampling time elapses and we can update state information by using the relevant equations. As is widely known, such a simulation policy is called *time-driven*.

On the other hand, other simulation systems do not need to support a continuous flow of time but are featured by events that can occur more or less sporadically. It is the case of a network scenario in which nodes may be silent for a long time and then make something happen by starting to send a packet. These scenarios are simulated by means of an *event-driven* approach: events are placed in a queue which is handled by a proper scheduler which retrieves and processes them accordingly. Events are also "timed", i.e. they have a timestamp indicating when that event must be processed: indeed, as an example, when a packet is sent, the next event is packet reception but it happens only after a certain time which depends on the transmission media and the packet size, a condition that a network simulation tool must consider.

As the reader can easily understand, our integrated simulation environment must deal with tools that have different simulation policies, namely time-driven for Gazebo and event-driven for ns-3. The solution we adopted to integrate them is to change the notion of time within ns-3 by writing a ns-3 module that provides an alternative event scheduler[7]: the new scheduler is able to receive a *target timestamp* from an external process, which triggers the execution of all scheduled network events whose

[7]The installation process simply consists in copying our module's source directory to the ns-3 directory. Simulation scripts can then make use of our module's API in order to tie ns-3's Nodes to Gazebo UAVs and High-level logic processes.

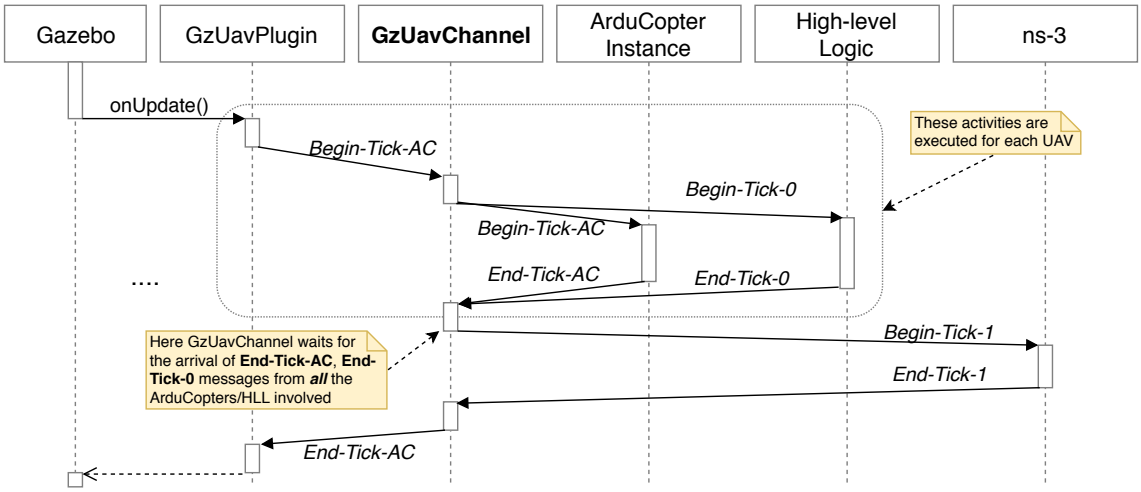


Figure 5.4: Sequence diagram showing messages exchanged among processes for each simulation step

timestamp is smaller than the received value. In this way, ns-3 becomes synchronized to an external clock source thus making integration possible.

A specific notion of time is not only required by the simulation tools but also by the other components of the framework. The ArduCopter flight stack is a time-driven process, and, since it implements the control algorithms for stabilization and navigation, it exploits differential equations too (that is, PID-based control schemes) which are thus discretised and implemented in a time-driven way. On the other hand, the notion of time for the High-level Logic depends on the kind of multi-UAV application to be implemented: strongly collaborative applications, like those based on flocking, are still based on algorithms requiring a discrete time step; other applications that require a loose coupling among UAVs are instead sensible to events such as the arrival of a network packet or the achievement of a certain way-point. In any case, for this component of the framework, a strict form of synchronization (with simulated time and with the other components) is required, too.

In our simulation environment, time synchronization is managed by GzUavChannel according to the interaction protocol that is depicted in Figure 5.4 and described here. Everything is started by Gazebo: it is this tool that indeed has the responsibility of generating the master time tick, an event that is notified to all the plugins running within Gazebo, including the GzUavPlugins. As Figure 5.4 shows, this event corresponds to the call to a specific callback function that, for the GzUavPlugin,

is called `onUpdate()`. As soon as a GzUavPlugin is notified of this event, it retrieves data relevant to the (simulated) sensors of the associated UAV (pose data), and sends them to GzUavChannel through a **Begin-Tick-AC** message. This message notifies GzUavChannel of the start of a new simulation period that is divided, by GzUavChannel, in two different sequential phases, called *Phase-0* and *Phase-1*: the first phase is devoted to the execution of two UAV-related activities, while, in the second phase, a step of network simulation is executed.

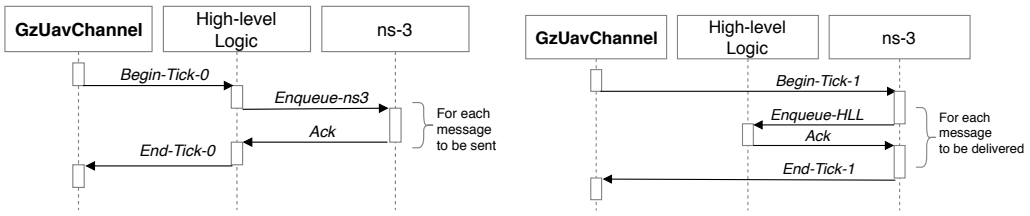
In particular, during Phase-0, as a consequence of the reception of each **Begin-Tick-AC** message from the plugin, GzUavChannel relays this message to the relevant ArduCopter process and, in parallel, generates another message, **Begin-Tick-0**, that is sent to the High-level Logic process. On this basis, the ArduCopter instance will execute one step of the control loop: the result will be the values of power to be sent to motors[8], an information that is replied to GzUavChannel by means of the message **End-Tick-AC**. In parallel, the High-level Logic, notified by the **Begin-Tick-0** message, executes both clock synchronisation and a step of its activities. Also this process, at the end of the step, replies to GzUavChannel by sending an **End-Tick-0**.

When all ArduCopter and High-level Logic processes have concluded their tasks and the GzUavChannel has gathered all the **End-Tick-AC** and **End-Tick-0** messages, the Phase-0 is over and *Phase-1* can be started. Phase-1 is characterised by the message **Begin-Tick-1** that GzUavChannel sends to the ns-3 process; as a consequence, ns-3 synchronises its clock and executes all the events whose timestamps fall within the duration of time tick. At the end of this step, ns-3 replies with a **End-Tick-1** message to GzUavChannel which, in turn, can signal the end of the simulation tick by sending a **End-Tick-AC** message to all the GzUavPlugins involved.

The subdivision of the simulation tick into two different phases is required to synchronise transmission activities. Indeed, during Phase-0, the High-level Logic can execute its simulation step of e.g. a flocking or other kind of algorithm; this phase surely would include the computation of next speed or position set-points for the UAV and their subsequent transmission to the flight stack (via MAVLink), as well as the transmission or reception of messages, via the wireless link, that must

[8]In particular, the output is the PWM values for the motor drivers.

be then simulated by ns-3. In order to correctly simulate network activities, the adopted policy is as follows: during Phase-0, High-level Logic processes that have to send data over the network directly interact with ns-3 by sending an **ENQUEUE-NS3** message, as Figure 5.5(a) details; here the message is not really processed by ns-3 but only placed in the event queue. When Phase-0 is over and Phase-1 is started, ns-3 scans the event queue and starts simulating data transmission: if, during time tick, a data packet needs to be delivered to a destination UAV, the relevant UavNode running within ns-3 is contacted which, on this basis, sends an **ENQUEUE-HLL** message (along with the packet payload data) to the High-level Logic process (see Figure 5.5(b)); once again, the packet is enqueued, and it will only be processed by the High-level Logic at the next instance of Phase-0.



((a)) Messages sent by UAVs are enqueued in ns-3 during Phase 0

((b)) Ns-3 runs the network simulation, including the delivery of messages to UAVs, during Phase 1

Figure 5.5: Interactions between High-level Logic processes and ns-3

5.5.4 Managing Simulations in a Distributed Environment

In addition to the synchronisation of the clock and the activities, the other aim of GzUavChannel is to allow a developer to distribute the simulation over different interconnected servers, in order to take advantage of a multi-node environment. In this way, the computational workload can be spread over a network, by e.g. partitioning the set of UAVs into a number of groups, each controlled by a different GzUavChannel instance on a dedicated computational node.

Basically, as Figure 5.3 shows, processes that interact directly, i.e. ArduCopter, High-level Logic and UavNode, are interconnected through a TCP Link that is independent of GzUavChannel, so they can be hosted on different machines. Nevertheless, all processes must be coordinated by the central entity that must do

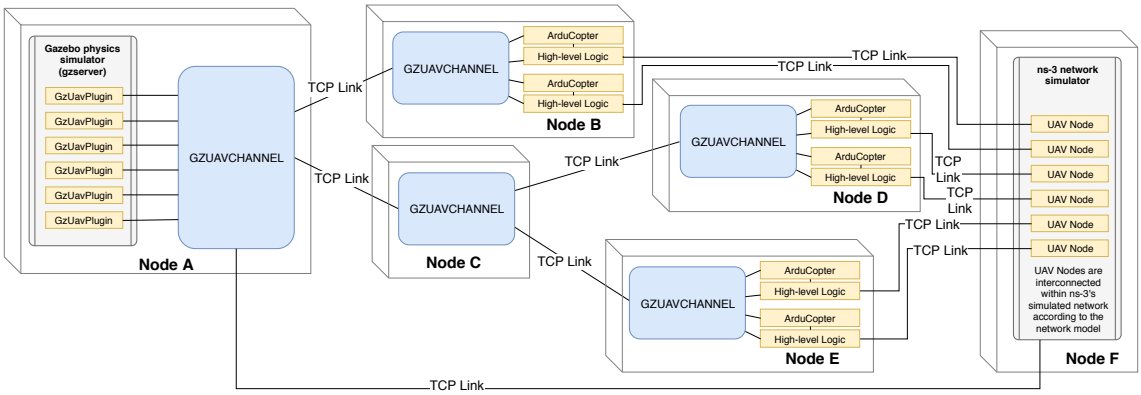


Figure 5.6: Architecture of the Integrated Simulator in a Distributed Environment

its work also when components are distributed, therefore a link to GzUavChannel must always be kept. Given that the policies used to spread the processes over a set of servers may be various, a good way to improve simulation performances is to separate the physics simulation (Gazebo) from the other parts.

In this sense, as Figure 5.6 reports, in the most general form, a tree can be created, in which the nodes are GzUavChannel instances and the edges are TCP connections[9]; in such a tree, the root is represented by the instance directly connected to Gazebo, while the leaves are connected to ArduCopter processes. Each ArduCopter process is connected to the associated High-Level logic process, running on the same node, which is in turn connected to the global ns-3 process.

The dynamics of interaction in a distributed environment are quite similar to the centralised case; coordination is performed by using the messages **Begin-Tick-AC** and **End-Tick-AC** that are exchanged by GzUavChannels. In detail, when the first **Begin-Tick-AC** is received from the root GzUavChannel (node A), it is forwarded to children nodes, down to the leaves of the tree. Each GzUavChannel thus behaves according to the protocol in Figure 5.4, coordinating the processes which it is has the responsibility for. Finally, when Phases-0 are completed, the **End-Tick-AC** message is generated by leaf nodes and forwarded along the tree until the root is reached, thus letting the root GzUavChannel proceed with Phase-1.

[9] As is widely known, TCP tends to introduce latencies that might slow down the simulation (but not affect its correctness, thanks to the strict synchronization protocol). The developed software tries to eliminate such latencies through careful usage of Linux's `TCP_NODELAY`, `TCP_CORK` and `MSG_MORE` flags in networking system calls.

5.6 Case-Study: Leader-Follower

In order to show how to use the integrated framework to develop and simulate a multi-UAV application, in this section we present a simple test-case in which two UAVs are involved in a *Leader-Follower* application. One of the UAVs (established at design time), the *Leader*, performs a mission on the basis of a list of fixed waypoints that are reached in sequence. At the same time, the current waypoint is sent through the wireless network to the other UAV, the *Follower*, which, as soon as it receives the packet, "follows" the leader by reaching the most recently received waypoint.

```
1  import math
2  import ns3interface
3  import simtime
4  # ... other imports
5  from dronekit import connect, VehicleMode, LocationGlobalRelative
6
7  ns3interface.connect(...) # Connect to the ns3 network simulator
8  vehicle = connect( ... ) # Connect to the Vehicle through MAVLink/DroneKit
9  simtime.connect(...) # Synchronize time.time() and time.sleep(n) with sim clock
10 while not vehicle.is_armable: # Don't try to arm until autopilot is ready
11     time.sleep(5)
12 vehicle.mode = VehicleMode("GUIDED") # Set flight mode to GUIDED and arm the copter
13 vehicle.armed = True
14 while not vehicle.armed: # Confirm vehicle armed before attempting to take off
15     time.sleep(1)
16 vehicle.simple_takeoff(5) # Take off to target altitude (5 meters)
17 time.sleep(5)             # wait for taking-off
18
19 # A square (in geographical coordinates)
20 WAYPOINTS = [ (-35.3631807, 149.1653119),
21               (-35.3633233, 149.1653333),
22               (-35.3633407, 149.1651585),
23               (-35.3631983, 149.1651369) ]
24 index = 0
25 while True:
26     target_lat, target_lon = WAYPOINTS[index]
   # Go to the waypoint at the current index
27     vehicle.simple_goto(LocationGlobalRelative(target_lat,
28     target_lon, target_altitude))
29     payload = struct.pack("<dd", target_lat,
30     target_lon) # Prepare the packet with waypoint
31     ns3interface.sendto(payload, follower_address)
   # Transmit the current waypoint
32     time.sleep(10)
33     index = (index + 1) % len(WAYPOINTS)              # Prepare for next waypoint
```

Figure 5.7: Simplified listing of the Leader

The simplified version of the source code for both the Leader and the Follower are reported in Figures 5.7 and 5.8, and described below.

After the necessary imports, both source codes feature a common preamble that includes all activities needed to initialize the communication channels (for this reason this common part is not reported in the listing of Figure 5.8 but simply referred to in the other listing). At first, such activities include the connection to ns-3 (line 7 of Figure 5.7), using our `ns3interface` library, and to the vehicle via DroneKit (line 8), and then the connection to GzUavChannel (line 9), using our `simtime` library; this last statement presents a library call that, together with the cited connection, redefines the functions `time()` and `sleep()` of the Python module `time` in order to allow a synchronisation of the process' time with the simulation tick[10]. The next activities of the preamble, carried out using the DroneKit library, are *(i)* ensuring that the vehicle is ready (lines 11-12); *(ii)* setting the vehicle in controlled mode[11] (lines 12-15); *(iii)* triggering the take-off at a certain altitude (lines 16-17).

As for the Leader (Figure 5.7), a list of waypoints is defined (lines 20-23) and then a loop is started including *(i)* request to reach the current waypoint (lines 26-27, through DroneKit); *(ii)* transmission of the current waypoint to the Follower (line 28-29, through `ns3interface`); *(iii)* waiting[12] (line 31, through the `simtime`-provided `sleep` function - which internally synchronises with GzUavChannel); *(iv)* change to the next waypoint in a circular fashion.

As for the Follower (Figure 5.8), it has the same preamble[13] (so it is not reported in the listing) and its main activity loop (lines 12-18) includes a check for the arrival of a new message: as soon as new data arrives through our `ns3interface` library (line 14), the packet is interpreted by extracting the coordinates of the waypoint (lines 15-16) and such a waypoint is set as the new target (line 17) using DroneKit. Idle time is spent by repeatedly calling `sleep`, which in turn causes our `simtime` module to let the simulation progress while keeping time synchronized with GzUavChannel.

[10] `simtime.connect` internally redefines such functions by simply assigning the alternate implementation to the function name (e.g. `time.sleep = new_sleep`). The redefinition happens at run-time and no permanent modifications to the Python libraries are required.

[11] This is called "GUIDED Mode" in the ArduCopter terminology.

[12] Indeed, at this point, instead of performing bare waiting, the program should retrieve current UAV position and check if the waypoint is reached; this can be performed by means of the API provided by DroneKit. However, for the sake of brevity, we omitted here this feature.

[13] The Follower's `target_altitude` must be different from the Leader's one to avoid collisions.

```
1  import math
2  import ns3interface
3  import simtime
4  # ... other imports
5  from dronekit import connect, VehicleMode, LocationGlobalRelative
6
7  #
8  # same initialization block of code of the Leader, lines 7-17
9  #
10
11 # We are the "follower" vehicle
12 while True:
13     # Process incoming messages
14     if ns3interface.message_available():
15         payload, sender = ns3interface.recvfrom()
   # Receive waypoint coordinates
16         target_lat, target_lon = struct.unpack("<dd", payload)
17         vehicle.simple_goto(LocationGlobalRelative(target_lat,
18         target_lon, target_altitude))
19     time.sleep(.1)  # Waste time if there is no available message
```

Figure 5.8: Simplified listing of the Follower

5.7 Performance Evaluation

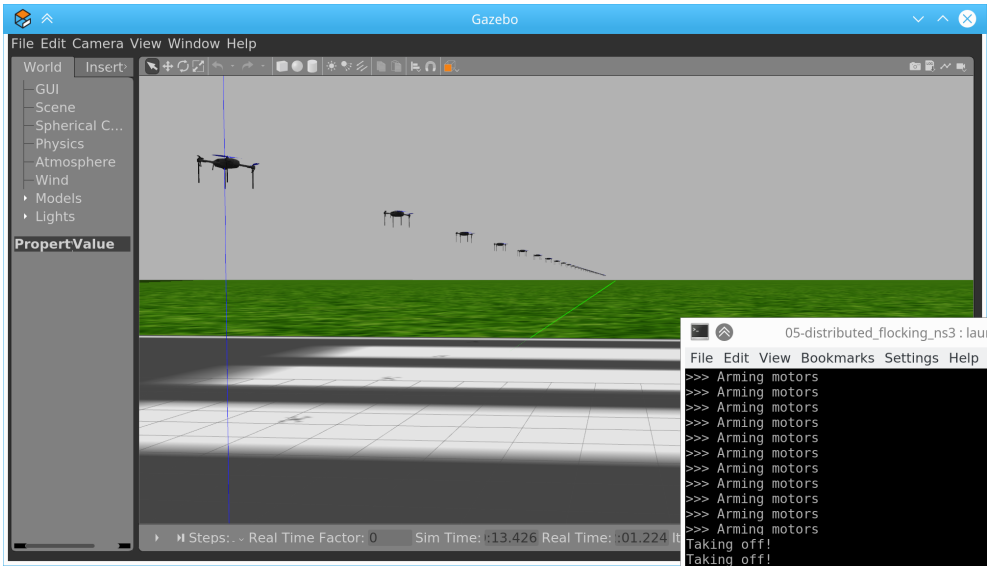


Figure 5.9: Screenshot of 40 UAVs taking off

The described architecture has been implemented and validated (see Figure 5.9) on a number of test programs. Among them, the implementation of the Boids algorithm [108] has been to evaluate the performances of the simulator. In particular,

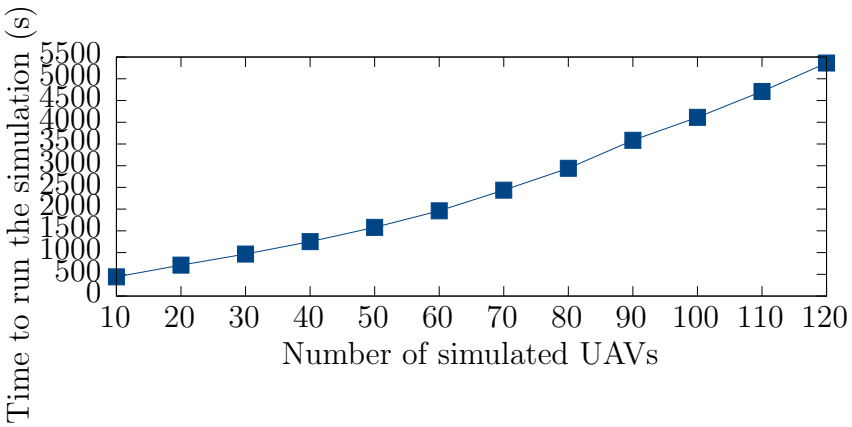


Figure 5.10: Simulation runtime corresponding to 280 seconds of simulated time

we aimed at testing *(i)* the scalability of the solution and *(ii)* its performances by comparing the duration of a simulation with respect to real-time.

We made a simulation campaign using an increasing number of UAVs starting from 10 up to 120, and using a (simulated) duration of the mission of 280 seconds. On this basis, we collected the measure of real-time duration of the simulation.

Measurements are reported in Figure 5.10, and refer to the implementation of the Boids algorithm that uses a IEEE 802.15.4 wireless channel for UAV interaction; the test run on a single Intel Xeon E5-2620 v3 @ 2.40GHz node, within a VMWare Virtual Machine with 8 vCPUs and 32 GB RAM. As Figure 5.10 shows, the integrated simulator is able to support up to more than one-hundred simulated UAVs, a number that is in accordance with a scenario of a real mission.

5.8 Conclusions

This chapter has described an architecture to combine different tools into a unified framework, which can be used to simulate multiple realistic UAVs with wireless networking capabilities.

Simulation is an invaluable tool in UAV application development, as it allows algorithms to be tested without using real UAVs, thus avoiding the associated risks. Thanks to the usage of ArduPilot's MAVLink API, the same code that runs on the simulator can also run in a real UAV. Therefore, our framework lets one develop

UAV programs in the simulator at first, and deploy them in the physical world only after they have been extensively tested and validated.

The choice of Gazebo, as the robotics simulation engine, makes it possible to include a variety of further objects and sensors in the simulated world for UAVs to interact with, in order to build realistic and accurate virtual environments. Gazebo also offers a 3D visualisation interface that allows one to optionally monitor the simulation progress in real time.

Multi-UAV applications are often strongly tied to the UAVs' ability to communicate with each other. Such communication requirements impose a careful performance analysis of available wireless protocols and technologies with respect to the specific application. Because of this need, we integrated the ns-3 network simulator in our framework. Ns-3 is able to accurately simulate most, if not all, technologies that are of interest for UAV applications.

The proposed framework combines Gazebo, ArduPilot and the ns-3 network simulator by means of a new tool we developed called GzUavChannel (and a set of adapter plug-ins for existing tools, too) which implements a custom protocol, to keep all parties synchronised. The result of this work is a time-based simulation framework, whose programs can be programmed in the Python language and then run, with minimal boilerplate changes, on real UAVs. Source code, as well as a number of example programs, can be found at the http://gzuav.dmi.unict.it/ website.

Future work will aim at implementing the flocking and area coverage algorithm described in [25], using the simulator during the development phase and then, once deployed on real UAVs, to analyse the differences between the simulator and the real world. We also plan to add support for unmanned ground vehicles (UGV), in order to support both UGV-only and UGV-UAV cooperation, as the framework is currently being used by our university's team in preparation for the MBZIRC 2020[14] robotics contest.

[14] Mohamed Bin Zayed International Robotics Challenge, http://www.mbzirc.com/challenge/2020

Chapter 6

Wale: libraries and packages sharing approach in Docker Containers

6.1 Introduction

Cloud Computing represents nowadays one of the most widely used enabling technology to access large computational resources [109]. The key aspect relies on the ability of the user of being connected to the Internet anytime and anywhere, thus allowing her/his to seamlessly access, with any device, to e.g. a remote disk space, an office application, a virtual CPU, or other kinds of computational resources. Services like Amazon Cloud, iCloud, Google Docs, Google Drive, etc., have become tools that people are normally using in everyday activities, both personal and professional.

Such a wide-spreading of cloud-based services implies the need of proper IT infrastructures able to support user requirements, not only in terms of response time and scalability, but also for what the privacy aspects are concerned. Accessing external storage or using network services often implies to send private data, e.g. documents, pictures, and any other kind of personal information, to remote servers that store or manage them, therefore the *secure* and *trusted* treatment of such data is an aspect that IT people must always strongly take into account.

From the technological point of view, the state-of-the-art enabling technology for Cloud Computing is *virtualisation* [110, 111, 112, 113], i.e. the ability to not directly expose, to users, a real CPU or a physical amount of RAM, but, by means of proper hardware and software tools, to "recreate" a virtual CPU, a logical/virtual RAM

space, etc. and let users/developers use them. Indeed, by means of proper virtualisers, a physical machine can be split into several *virtual machines (VMs)*, each one with its proper characteristics derived on the basis of the particular service that the VM itself has to support or provide. There are many off-the-shelf solutions that support virtualisation; many of them, i.e. hypervisors [114, 115, 116], exploit dedicated hardware, but there are also software-only solutions that, while being somewhat limited, perform quite well for certain specific types of cloud environments.

Among software-only solutions, **Docker** [117] represents one step beyond: Docker allows operators to create execution environments without using any kind of software or hardware emulation. Docker has recently gained a lot of attention in the context of *DevOps* [118], as the enabling technology to easily create and deploy virtual execution environment suitable for *microservices* [119, 120], which are quite hard to support using traditional (hardware-based) virtualisation approaches. Docker relies on the use of so called "container images", i.e. complete installations of a Linux-based execution environment which can be created from a build file or downloaded from an official or third-party repository. A single physical machine can host multiple Docker containers, thus providing different virtual environments, each one running its OS image. These images represent ready-to-use environments suitable to deploy applications ready to be executed in containers. In this way, developers can publish ready-to-use images for every kind of application and service.

In addition to Cloud Computing, developing of new kinds of Edge Computing and IoT infrastructures are made possible thanks to Docker. Indeed, a microservice can be deployed just-in-time on some nodes closer (in terms of network hops) to the node requiring that service.

Nowadays, these infrastructures have found a large number of applications, from the management of energy saving (through on-site control of energy production with appropriate devices) to the medical environment (via mobile devices worn for example by patients)[121, 122, 123, 124], bringing us closer to Tactile Internet (chapter 2) [19].

Unfortunately, despite the cited advantages, Docker presents an important drawback: building and deploying Docker images might generate non-trivial disk space waste, which is caused by the large number of libraries that each application needs

for its execution. Furthermore, since containers represent isolated environments, every library or package used in more than one container will be duplicated, because there is no sharing mechanism. A consequence of this is a waste of disk space, that increases on the basis of the number of containers running on the physical machine and the amount and size of common parts.

In this context, this chapter proposes a smart image building management technique, for the Docker platform, which allows developers to manage the creation of the various containers images by including common libraries into a base docker image called *Core Image*. The developed technique allows developers and system administrators to save space on the disk without affecting the organisation of the application containers. The technique, called *Wale*, is based on the sharing of the libraries and packages that could belong to more than a container, but without affecting the requirements of privacy that completely isolated environments provide by default. A case study is included in the chapter, with some results proving that the proposed approach is effective in terms of saved space in container management.

6.2 Related works

In the wide-spreading of the virtualization and containerization technologies such as Docker, de-duplication of memory and disk space is becoming an important non-functional requirement for enterprises which have moved their computational infrastructures in the cloud. Although our work is focused on avoiding duplication of disk space, a few related works are similarly aimed at saving memory [125, 126, 127, 128].

Among them, KSM (Kernel Samepage Merging) [126] represents a technique developed to prevent RAM duplication in the same host across different virtual machines which possibly runs the same software or handle the same data. KSM is simply a Linux Kernel module to enable sharing of parts of memory across different processes and, as a consequence, also for different KVM (Kernel based Virtual Machine) virtual machines. The main advantage of KSM is that it finds memory pages with the same content in the system by using two different data structures: it uses two trees, one representing the already shared and not changing KSM generated pages (the “stable tree”), the other representing the memory pages which

are not shared but that are tracked by KSM. However, due to efficiency issues, KSM provides a set of API to let the applications optionally register which virtual memory areas should be scanned by the kernel thread that has the task of merging equal physical pages of memory. Moreover, some possible security issues have been analysed. As discussed in [129], memory de-duplication is vulnerable to memory disclosure attacks. Since de-duplication is performed by the Copy-On-Write technique, the authors observe that, although the sequence of operations is logically valid and the results is consistent, the write access time is different between de-duplicated and non-de-duplicated pages. As a consequence, an attacker may use the time difference in a memory disclosure attack.

The researchers studied the problem of disk de-duplication since the beginning of the massive adoption of virtualisation [130]. The authors of [130], in order to test effectiveness of de-duplication, conducted a number of extensive evaluations on virtual machine disk images with different "chunking" strategies. They have shown that de-duplication of VM disk images can save 80% or more of the space required to store the operating system and application environment. They found that the de-duplication ratio can have a high bias due to the many factors such as the base operating system or even the Linux distribution. Overall, the tests performed by the authors have shown that, in general, 40% is approximately the highest de-duplication ratio if no obviously similar VMs are involved (e.g. same OS selected by the users).

A project worth to mention is LCFS[1], Layer Cloning FileSystem, which is a filesystem designed to be a Docker storage driver[2]. LCFS focuses on image's layers, and it operates directly on top of block devices, as opposed to merging separate filesystems. Thereby, LCFS aims to directly manage the container image's layer level, which has a positive impact on the overhead of having a second filesystem.

File system block level can provide several advantages. Docker provides several different storage drivers, which are supported by different backing filesystems. In particular, as widely documented[3], `aufs`, `overlay`, and `overlay2` operate at the file level rather than the block level. The main implication is that they are efficient in terms of memory consumption, but there is no de-duplication of disk space, and the container's layer may significantly grow in write-heavy workloads. The three

[1]https://github.com/portworx/lcfs
[2]https://docs.docker.com/storage/storagedriver/select-storage-driver/
[3]https://docs.docker.com/storage/storagedriver/select-storage-driver/

aforementioned storage drivers are supported by `ext4` or `xfs` filesystem. Instead, `btrfs` or `zfs` storage drivers, supported by the backing filesystem which have the same name, BTRFS [131] and ZFS [132, 133], operate at block level. This choice leads to a better efficiency, as they work at the block level. Moreover, both zfs and btrfs support de-duplication at block level. On one hand, if storage de-duplication is performed at block level, it is transparent to the operating system, Docker and all the applications. On the other hand, operating such a choice imposes the adoption of one of the filesystem zfs or btrfs, as ext4 or xfs can only support overlay/overlay2 and aufs. Finally both btrfs and zfs are less efficient in terms of memory consumption.

6.3 Background

6.3.1 Virtualisation via Hypervisors

As it has been stated in Section 6.1, *virtualisation* [110] is the key technology to make a physical host machine able to emulate a number of *virtual machines (VMs)* with, at most, the same hardware specifications of the host machine itself [113]. The main component for virtualisation is the *Hypervisor*, also known as *Virtual Machine Monitor (VMM)*[4]; it is a software, installed on top of computer hardware, that creates a virtualisation layer with the objective of managing the sharing of physical resources. The VMM emulates the hardware of a physical machine and prevents access, by a VM, to the real hardware thus providing an isolation environment ensuring that a VM cannot affect the operations of any other VM and host system, even in the case of crashes or malicious behaviors. The way in which this task is performed is a crucial aspect, since it is main impact factor of VMs performance. In this sense, two kind of VMM technology exist, known as *type-1* and *type-2*. Type-1 hypervisors, also known as *bare metal hypervisors* (see left-side of Figure 6.1), are small executives that boot when the machine is powered-up; in turn, they can run the guest OS of each virtual machine as a user process of the executive. The guest OS is unaware that it is just a user process of the hypervisor and when it tries to execute a sensitive instruction that could imply the access to a hardware resource, a

[4]In the following, we will use the terms "hypervisor" and "VMM" with the same meaning.

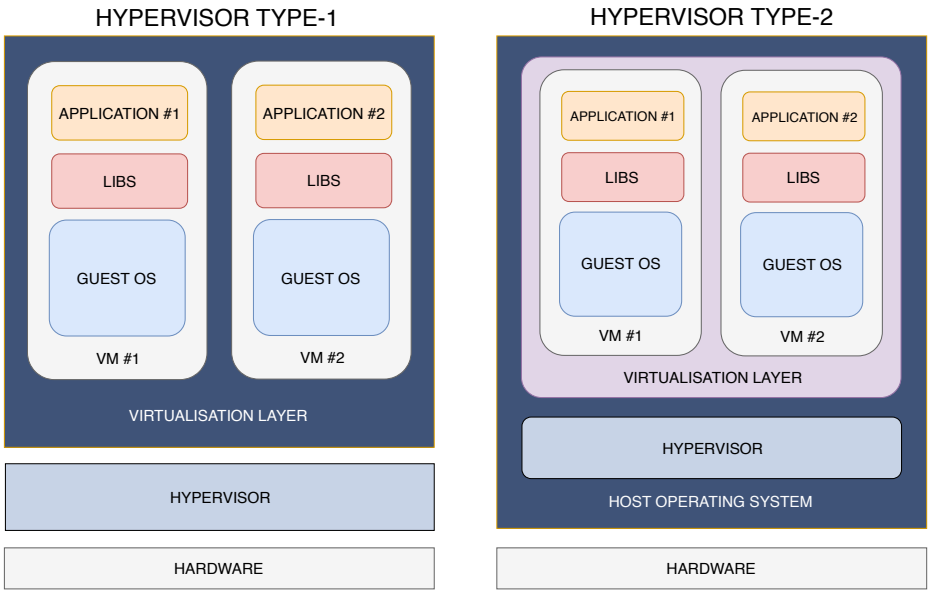


Figure 6.1: Hypervisors Type-1 and Type-2

trap to the hypervisor occurs that, in turn, manages it accordingly. Xen [110, 134], ESXi [135, 136, 137] and Hyper-V [136, 137] are examples of type-1 hypervisors.

Type-2 hypervisors, also known as *hosted hypervisors*, are software applications installed and running directly on top of a host operating system (see right-side of Figure 6.1). To support virtualisation, this kind of hypervisors either use the CPU capability to intercept sensitive instructions, which are then processed by an agent running in the host's kernel, or use the "binary translation" [138, 139, 140] technique: the guest OS (as well as any of its software applications) is scanned by the hypervisor in order to search for code blocks containing sensitive (i.e. reserved) instructions, such parts are then replaced to calls to the VMM in order to be emulated and handled properly. VMWare [141, 142], VirtualBox [142], Bochs [143, 144], KVM [135, 137] and QEMU [145] are instances of type-2 hypervisors.

6.3.2 The Docker Approach

The philosophy behind Docker [117] is instead completely different with respect to the solutions cited above. Unlike hypervisors, Docker is able to create virtual environments that share *the same* Linux kernel: as Figure 6.1 shows, the idea is to move the virtualisation layer up to the application space (as in type-2 VMMs) but,

instead of having a different kernel for each VM, all VMs share the same kernel. This solution surely makes it impossible to organise VMs with different guest OSs, but, when the same OS suffices for the target application, this approach surely provides less overhead with respect to hypervisors[5].

In Docker, the term "virtual machine" is replaced by a more generic "virtual environment" which is called *container*: indeed, a Docker installation is a set of *containers*, each one properly isolated just like a VM. Docker containers are managed by the *Docker Daemon*, a background service that runs on top of the OS and manages the creation, termination and execution of the various defined containers.

Virtualisation and isolation are achieved in various ways. File system isolation is obtained by exploiting the "chroot" tool[6], while the Union File System (UnionFS) [146] is exploited to handle file system changes in order to *(i)* ensure that each container has its own files, *(ii)* undo changes when the container is destroyed. UnionFS is based on a layering approach: once the Docker Daemon launches a container, it mounts the root file system in read-only mode and another layer is added to the file system; every time a file system change occurs, Docker adds a new layer. Cancelling changes thus implies only to remove layers.

From the practical point of view, each container is made of a special package file, called *Docker Image*, which includes all the libraries and applications needed for that container and that is created by a compiler able to process a text file that specifies the composition of the image, in terms of additional software packages, libraries and applications. Creating a Docker container implies (for the Docker Daemon) to instantiate the relevant image file thus creating the virtual environment. The details of the process for compiling and creating a Docker Image, as well as the syntax and semantics of the image specification file, are given in the next Section.

6.4 Docker Images and Dockerfiles

As it has been reported in the previous Section, Docker uses UnionFS [146] to manage the Docker images, a service that is able to mount files and directories from

[5]Indeed, using hypervisors, we need a complete OS installation for each VMs.
[6]http://www.gnu.org/software/coreutils/manual/html_node/chroot-invocation.html

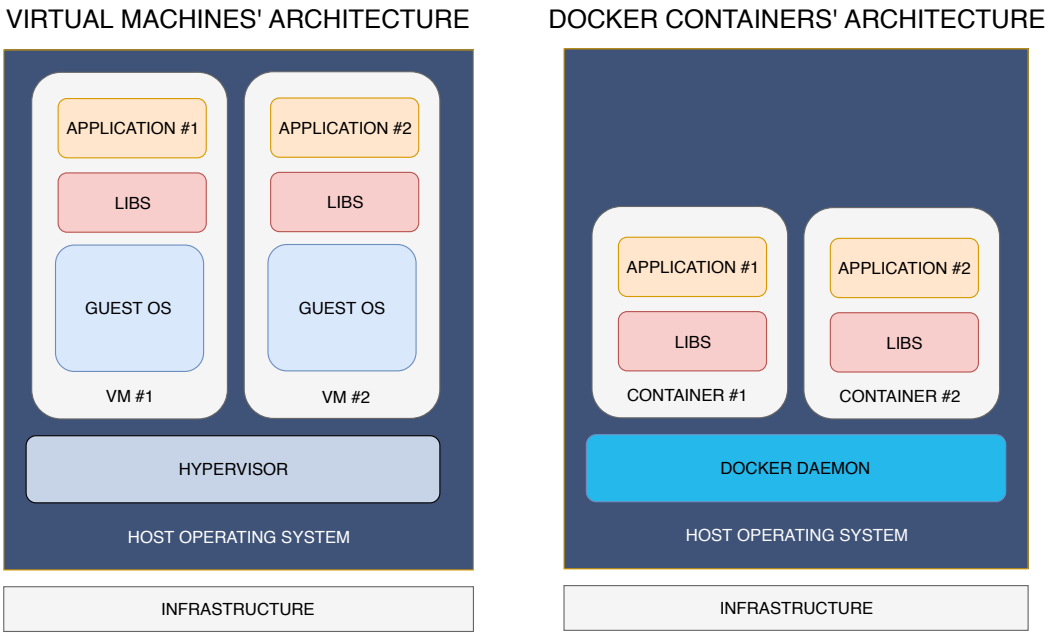


Figure 6.2: Virtual machines and Docker Containers architectures

other file systems and combine them into a single file system; indeed, a Docker Image typically contains a union of layered filesystems, stacked on top of each other.

A Docker Image is made starting from a *base image*, which includes the *root filesystem* of the Linux distribution to be used in the container, plus the files of the application(s) to be run; moreover, a series of packages can be added, if they are needed by the applications themselves. All modifications to be applied to the base image to obtain the final Docker Image are specified by using a text file, called "Dockerfile", which includes a set of directives following a precise syntax[7], an example of which is reported in Listing 6.1. As the listing shows, the "`FROM`" directive specifies the root file system to be used for the base image; the "`ADD`" directive indicates an application file to be added, while the "`RUN`" directive implies the execution of a specific command that, usually, is employed to install additional libraries or packages; finally, the "`ENTRYPOINT`" directive is used to run the application when the container is instantiated.

A Dockerfile is parsed by a proper compiler that generates the Docker image used to instantiate the container. This last operation (i.e. launching a new container), which is performed by the Docker daemon, implies a series of operations that follow

[7]https://docs.docker.com/engine/reference/builder/

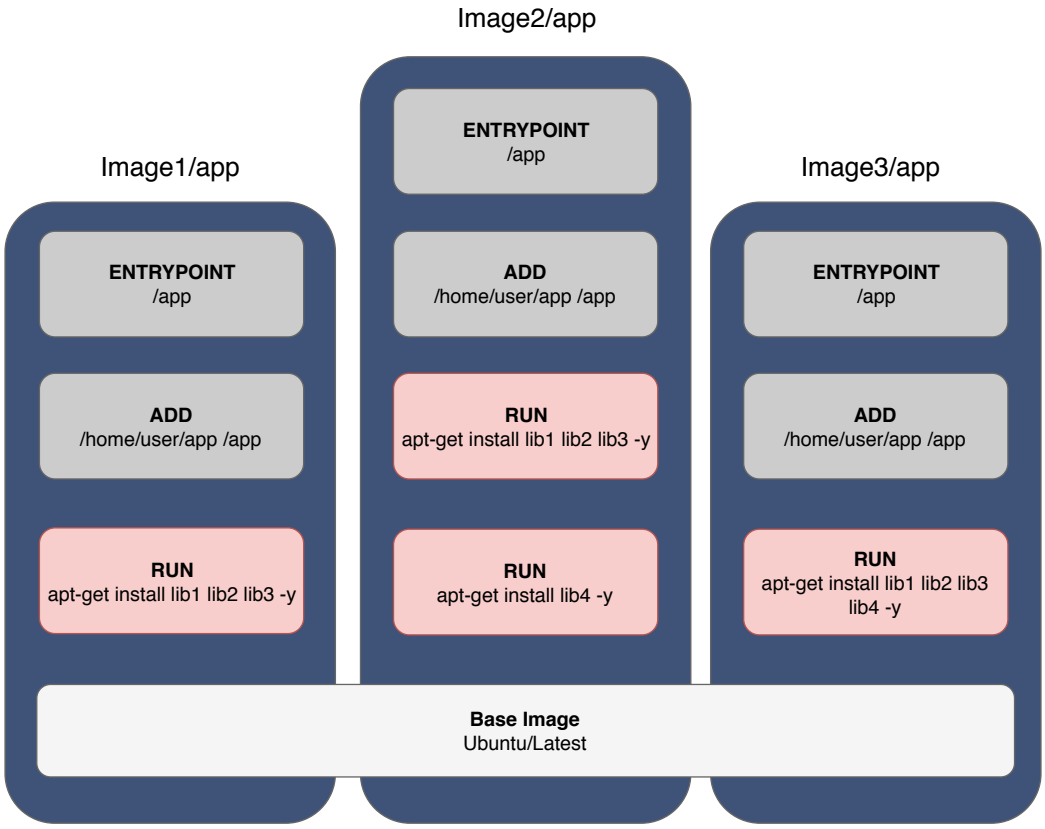


Figure 6.3: Three Docker images with same root file system, libraries and application

the content of the Dockerfile and that are detailed in the following. First the root file system of the base image is mounted in read-only mode; then, in order to support file system modification, another layer is added using UnionFS mount. According to the Dockerfile, the various lines are thus "executed" in order, but, for each line, a new (UnionFS) layer is added; this process is shown in Figure 6.3.

When different containers are instantiated in the same machine, a sharing policy is applied to the base image only: if more than one container uses the same root file system image, Docker organises them in order to share the content (see Figure 6.3); this is made possible thanks to the fact that the root file system is mounted in read-only and that modifications, in the specific containers, are handled by using layering; therefore, a change done in a container is not propagated in/visible by other containers. Since it is very common to have different applications (running in different containers) that however are based on the same running platform, this sharing policy makes sense because it is able to save disk space.

Listing 6.1: Example of Dockerfile

```
FROM ubuntu:16.04
WORKDIR /app
RUN apt-get install node -y
ADD package.json package.json
ADD main.js main.js
RUN npm install
ENTRYPOINT node main.js
```

Unfortunately, the sharing is applied only to root filesystem image and not to other parts, such as libraries or applications files; in many cases, the same library (or even a same set of libraries) could be needed by different containers; and, even more, many people use Dockerfiles made by third parties, without paying attention to their content, just to build the image and run its Docker container. In all of such cases, the lack of a sharing policy at the library/application level surely leads to a wasting of disk space, that can also be quite relevant when the number of containers increases. The solution we developed is able to address this problem thus allowing developers to save disk space without affecting the isolation degree needed in a Docker environment.

6.5 The Wale approach

6.5.1 Basic Working Principle

As it has been already discussed in Section 6.4, a Docker Image starts from a base image and includes a set of libraries and some specific application files. According to Figure 6.3, the base image is shared among all containers and thus can be considered somewhat "public" since it is visible from any created virtual environment, even if in read-only mode. All the other parts, i.e. libraries and application, are instead container-local and thus completely isolated from the relevant parts of other containers. We can refer to them as "private" parts.

This subdivision into public and private parts is embedded in Docker by design. Indeed, the underlying concept of Docker is to create environments that are isolated from each other, such as "sandboxes", so that these environments can not affect to

each others, otherwise the concerns of privacy and security, that are fundamental for cloud environments, would be lacking. Nevertheless, it is plausible to think that a library or package used by two or more containers could be shared as well, in order to save space on disk that would otherwise be wasted. However, even if, from a point of view, sharing a common library implies to save disk space, from another perspective it must not compromise–in any case–the isolation, privacy and integrity of containers.

The approach proposed in Wale[8] is based on the concepts above. The idea behind Wale is, at first sight, simple but effective: to identify all libraries/files that are common to several containers and move them from the "private" parts (that would feature duplication) to the "public" part (that features sharing). The idea is therefore to exploit this public/private mechanism of Docker to avoid the waste of disk space, being able to identify what can be transported in the public part and what should remain in the private and isolated part. By applying this solution, the structure of the three containers depicted in Figure 6.3 would become as in Figure 6.4: here the commands relevant to the installation of packages are executed in such a way as to download and install files in the public part, so that they can be shared by any container.

Of course, placing packages in the public part must be somewhat controlled and limited, otherwise the requirements of privacy and isolation could be easily violated: indeed, it's up to the Wale tool to organise and properly decide which parts can be moved to the public part by following precise rules. These aspects, as well as the details about the working model of Wale, are reported in the next subsections.

6.5.2 The Wale Tool

In order to support the sharing characteristics explained above, the Wale tool acts as a pre-processor by taking, as input, a file specifying the desired composition of a container and generating, as output, a Docker Image that can be directly instantiated by Docker itself. With reference to Figure 6.4, the public part is called *Core Image* (which conceptually replaces the base image of the original Docker structure). Like the private part, the Core Image is managed by UnionFS and thus is organised in "layers": the lower layer corresponds to a Docker base image

[8]The name was chosen because it sounds similar to the word "whale".

Algorithm 1 Wale's approach algorithm

```
1: procedure WALE(core, application)                          ▷ Dockerfiles
2:     depends ← GetDependencies(application)
3:     n_depends ← depend.length
4:     if core not exists then
5:         core ← GetFromDockerHub(application.requiredDistribution)
6:     end if
7:     if n_depends > 0 then
8:         core_depends ← GetDependencies(core)
9:         diff_depends ← core_depends − depends
10:        n_diff_depends ← diff_depends.length
11:        if n_diff_depends > 0 then
12:            for dependency in diff_depends do
13:                AddDependency(dependency, core)
14:            end for
15:            BuildImage(core)
16:            aux_app ← CreateAux(application)
17:        end if
18:    else
19:        aux_app ← CreateAux(application)
20:    end if
21:    BuildImage(aux_app)
22: end procedure
```

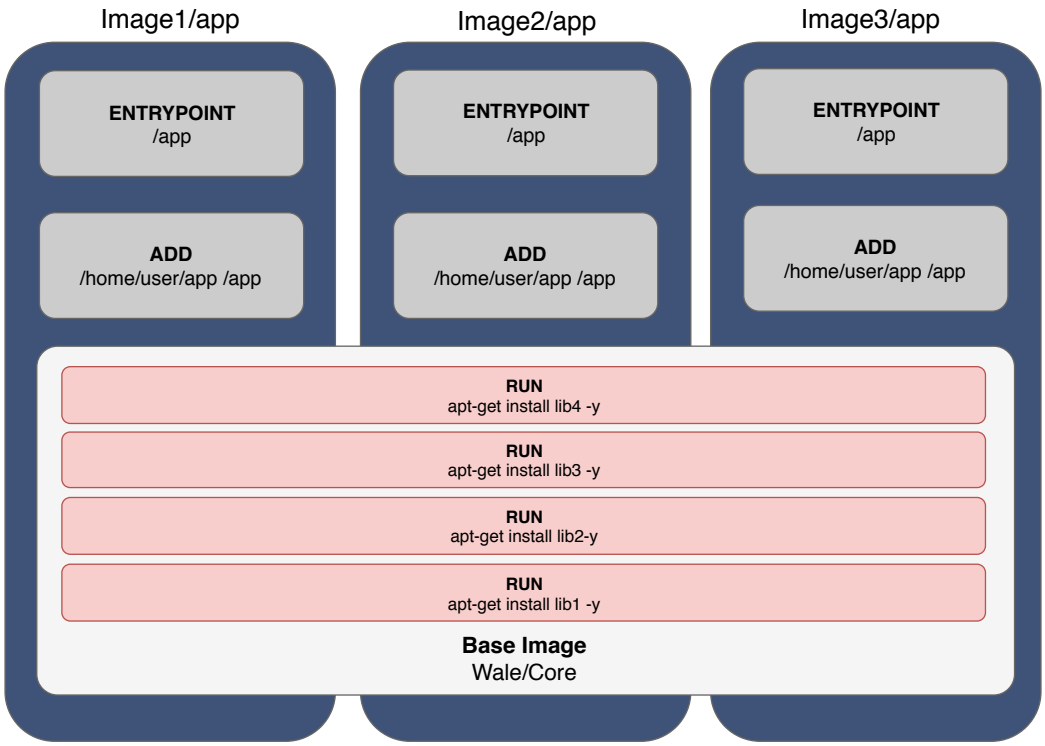


Figure 6.4: Three Application images with same base image (Core image) and then same libreries

(e.g. Debian-based or a Red Hat distribution), while each additional layer is created for each additional package installed. Wale takes care of managing and checking, with appropriate specifications, what has to be installed inside the Core Image and, later, how to create a new application image starting from the Core image, but inserting all the private information related to the application and the application itself.

The working scheme of Wale is shown in Figure 6.5. Everything starts from a text file, called *Wale file*, which describes with a JSON syntax the container to be created; this file specifies the following information:

- **distribution:** it indicates the base image and is equivalent to the "`FROM`" keyword of a Dockerfile;
- **dependencies:** it is a list of additional packages that need to be installed (via "`apt-get`" or a similar command);

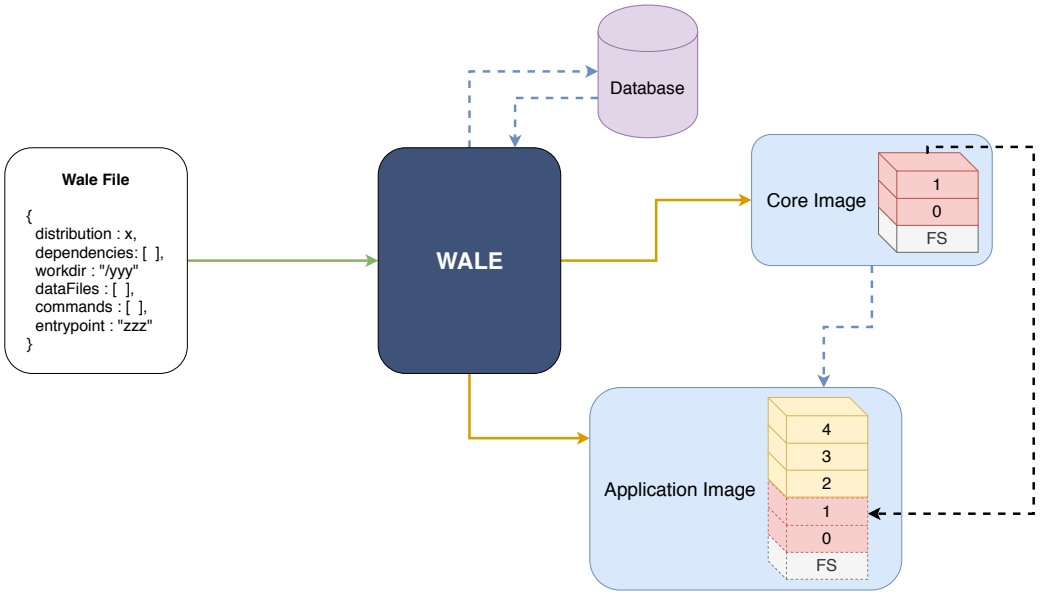


Figure 6.5: The Work-flow of Wale

- **dataFiles:** it is a list of application/data files to be copied in the final image that are needed by the application to be run in the container;
- **commands:** it is a list of commands that are specifically related to the application and that must be run each time the application image is built;
- **workdir:** it specifies the directory in which the files of the application will reside;
- **entrypoint:** it specifies the shell command to execute in order to lauch the application in the container.

When the container will be instantiated, the first two items will be placed to the public/shared part, while the remaining items will be placed/run in the private part.

In order to create and instantiate the container, the text file is parsed by the Wale tool and the various parts specified are handled according to Algorithm 1 which is described below.

First of all Wale checks whether, in its database, a Core Image corresponding to to the distribution base image already exists; if this is not the case a new Core Image is created by Wale (lines 4–6 of Algorithm 1). In the retrieved Core Image, layers correspond to the various installed packages: Wale compares the list of installed

packages with the packages requested by the new specification file (i.e. "dependencies") (lines 8–10) and missing packages are added by stacking new layers (lines 11–15). The process of building the Core Image is performed, by Wale, by exploiting the tools of Docker: the aim is to create a new Dockerfile, representing the new Core Image, and compile it; however, this task is managed by Wale and thus completely transparent for the user.

In a way similar to the creation of the Core Image, the sections of the Wale file related to "dataFiles", "commands", "workdir" and "entrypoint" are subsequently processed by Wale: the result is another Dockerfile that imports the Core Image above and includes the proper commands (lines 16–19). When this Dockerfile is compiled by the Docker tool, the final image of the container is obtained and can be thus directly instantiated (line 21).

6.5.3 Example of a Wale file

Listing 6.2: Wale file

```
1 {
2     distribution : "ubuntu:16.04",
3     dependencies : [ "node" ],
4     dataFiles : [ "package.json", "main.js" ],
5     commands : [ "npm install" ],
6     workdir : "/app",
7     entrypoint : "node main.js"
8 }
```

As an example, Listing 6.2 reports a Wale file for the creation of a container running a specific NodeJS application. The container uses an Ubuntu 16.04 distribution and requires the "node" package to be installed; moreover, the files "package.json" and "main.js" must be included[9], the command "npm install" must be executed before running the application, the directory of the application has to be "/app" and the application itself must finally be launched by means of the command "node main.js".

[9]They constitute the application files so they are provided separately.

Starting from this file, Wale retrieves the Core Image relevant to Ubuntu 16.04[10], along with the associated Dockerfile, and modifies the latter by including the lines relevant to the additional packages requested. The result is a new Dockerfile, shown in Listing 6.3, that, when compiled, produces a new version of the Core Image relevant to the chosen distribution. The Core image Dockerfile acts as "database" for Wale tool. In fact, this Dockerfile describes a list of already installed libraries and dependencies. In addiction, this Dockerfile can be used to backup and re-build the Core image.

Listing 6.3: Rebuild of the core image

```
1 FROM ubuntu:16.04
2 #DEPENDENCIES
3 ....
4 ....
5 RUN apt-get install node -y
6 #END_DEPENDENCIES
```

The other parts of the Wale file are finally used to create the application Dockerfile, which is depicted in Listing 6.4, that, compiled with the Docker tool, is able to create the final desired image of the container.

[10]We suppose that this image is present in the database.

Listing 6.4: Re-build application image

```
1 FROM wale/core
2 WORKDIR /app
3 ADD package.json package.json
4 ADD main.js main.js
5 RUN npm install
6 ENTRYPOINT node main.js
```

6.5.4 Images Deletion and Garbage Collection

The working scheme of Wale described so far is related to the creation and instantiation of container images built according to certain specification files. However, during working sessions, images are not only created but also deleted when they are no more needed: in such a case, all packages that are used by a deleted image should be removed, otherwise they will waste disk space, but such a deletion must be performed *only if* the packages are not shared/used by other application images. For these reasons, deletion is a process that has to be managed by Wale by means of a two-fold mechanism: *(i)* the use of a tag based on reference counting, and *(ii)* by running a garbage collection process. Wale includes an additional database that stores the list of packages which are installed into the Core Image and tags each package with a *reference counter*. Such a reference counter aims to count how many Application Images are using that package, i.e. it is incremented each time the package is installed into a new Core Image, and decremented when an Application Image is deleted: a reference counter equal to zero thus means that the package is no longer used by any Application Image and can be deleted from the Core image(s). This operation is handled by Wale tool[11] which scans the reference database in order to find packages with counter equal to zero and rebuilds the relevant Core Images accordingly.

6.5.5 Isolation and Privacy of containers

The objective of Wale is to support library sharing among Docker containers in order to save disk space. At first sight, the presence of additional shared parts could lead

[11] that must be run each time an Application Image is deleted

to think that the isolation requirement is compromised. However, despite the goal achieved, the final isolation is not lacking.

First of all, Wale is an approach that allows the management of the compilation of Docker Images through the simple management and updating of the related Dockerfiles, without therefore altering the normal operation of the Docker Daemon, so the same isolation among multiple containers that Docker guarantees is kept intact in the Wale context.

Secondly, the Wale building phase forces a developer to use images from the official Docker repository (that is called Docker Hub); for this reason the risk to use images which can be (maliciously) altered is completely avoided.

The guarantee of the Docker container isolation does not imply the total privacy between the containers: indeed, if used incorrectly, Wale may install malicious libraries within the same Core and then share them in all containers, effectively infecting them. To overcome this problem, an additional restriction has been added in Wale: a developer is allowed to use exclusively the `RUN` commands with the correlated package manager to install packages in the shared dependency block. With this additional Wale constraint, the risk of including malicious code, that can alter several containers, is thus avoided: code that becomes part of the Core can be installed exclusively from the package manager (e.g. Aptitude, via the `RUN` command) by using trusted repositories that are specified in Docker images and that cannot be modified by the Wale user; on the other hand, other third-part (untrusted) libraries– if needed–can be only included in the local context of a certain container and it's up to the developer itself the responsibility of their trustiness: nevertheless, such additional libraries could contain malicious code, but it affects only the specific container and not the whole Core.

6.6 Case Study and Experimental Data

In order to validate the approach proposed in this work, we describe, in this Section, a case-study of Wale used in a desktop application environment thus reporting the performances measured in terms of disk space utilisation. Our experiment was performed in a Linux Mint distribution (18.3 Sylvia) using Docker Community Edition

(18.05). The hardware details are omitted because we only evaluate the disk space usage.

Although Docker is mostly used for deploying server-side applications, it can be used to run desktop applications. Moreover, a desktop environment is more appropriate for our purposes since it is a scenario in which the advantages of Wale are more evident.

As applications, we have chosen some of the most commonly used ones, that are: Atom, Chrome, Firefox, Telegram and Visual Studio Code. These applications share the most used graphics libraries, such as libcairo2, libgl1-mesa-glx, libgl1-mesa-dri, libx11-xcb-dev, etc. with all their dependencies.

In this experiment, we have been used desktop applications (with GUIs) only to better highlight, in terms of used disk space, the capabilities of Wale to reduce the disk space that can be wasted, because the graphical libraries could be more heavy than other libraries. Nevertheless, the results obtained also extend to services and microservices. Once we prepared the various Dockerfiles[12], we started the build phase[13]. In our experiment, the various Docker images used two different root file systems: **debian:sid** and **debian:buster**. In order to render the Graphical User Interface that a desktop application could have, we mount the X11 socket into the container and we define the device where we want redirect the out rendering (typically the first/unique display). We run the created images and measured the resulting disk space: we noticed that the storage used by these applications was much more than their official storage requirements. Indeed, applications such as Chrome or Atom reach almost 900 Megabytes of disk space. Overall, the entire installation occupies 2958 Megabytes. These numbers are reported in Figure 6.6 and Table 6.1.

We repeated the same experiment by using Wale, by properly preparing the various Wale files of the applications and running the Wale tool accordingly. The results, also shown in Figure 6.6 and Table 6.1, proves that, in most cases (Chrome, Firefox and VSCode), the amount of saved space is more than 50 % with respect to the corresponding Docker build, while, in average, the disk space occupancy of reduced to the 40.91 % (1210.1 Megabytes).

[12]In this preparation, we exploited some Dockerfiles ready to be downloaded from the Internet.

[13]The original used Dockerfiles were created by the developer Jessie Franzelle and are available on her git repository *https://github.com/jessfraz/dockerfiles*.

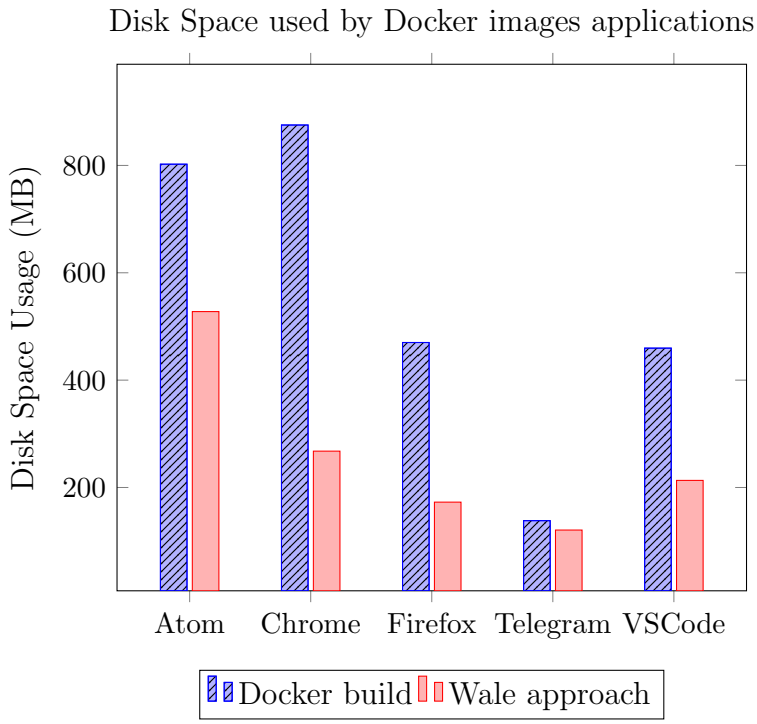


Figure 6.6: Disk space usage for different applications

As a final remark, we have to consider that the percentage could increase depending on the number of the applications and services, because the more amount of shared parts the more the disk space saved.

6.6.1 Discussion

The Wale Approach, as already described above, allows a developer to save disk space by sharing, among several images, some of the common libraries. It is a non-invasive approach to the Docker technology as it defines a method to build images, in line with current technology. In fact, the Wale approach can be directly implemented in any machine having Docker installed. Wale does not alter Docker in any way, in fact isolation among containers is still guaranteed. In other words, each container will not be able to alter other library applications.

However, despite the cite advantages, Wale presents a drawback that is related to application/libraries updating. Indeed, when an application receives an update (e.g., a major or a security update) all images that use it should be rebuild: this

Disk Space Usage

Image	Docker build (MB)	Wale approach (MB)
Core Image		445.9
Debian:sid	105.9	
Debian:buster	106.6	
Atom	802.3	527.7
Chrome	875.3	267.7
Firefox	470.1	172.7
Telegram	138.2	120.7
VS Code	459.6	213.2
Total	2958	1747,9

Table 6.1: Disk space usage for different applications.

operation can be performed in two different ways. The first one concerns installing the updated library into each application image, which surely means a waste of disk space; for this reason, this method is advised only if a few images require this updated library. The second way, that can be used when disk space needs to be saved, implies to install the updated library into the Core Image; more specifically, this method requires to rebuild both the Core Image and all images that are using this updated library. In both cases, a non-negligible cost has to be payed, since one or more images need to be rebuilt, however, since major updates are not so frequent, such a cost cannot be considered particularly high with respect to the advantages, in terms of disk space saving, that the Wale approach is able to provide.

6.7 Conclusions

Docker is a system that has revolutionised and is revolutionising the way in which applications and services are deployed. Being able to manage and manipulate, in a clear and simplified way, a complex ecosystem of microservices within a cloud infrastructure brings great benefits, not only in terms of timing but also in terms of security. Docker allows developers to create contained environments within which to manage applications in isolation. Unfortunately, Docker does not take into account the space occupied by every single image. This is due to the fact that natively it

was not designed to keep track of what is installed but exclusively to create isolated environments.

To overcome such a limitation, in this chapter, we presented a solution based on a simple image build approach that saves disk space by sharing common libraries among multiple containers. The proposed solution is based on a tool, called Wale, that has the task of automating the creation of Dockerfiles and the relevant images, being able to recognise the parts that can be shared among different containers. Adopting this solution leaves the host unchanged and it does not require the adoption of a specific file system in the host or in the VM.

As future work, we aim at testing and/or integrating the Wale mechanisms as a module of the Moby Project [14], which is a recent project with the objective of building stratified modules for Docker.

[14] https://mobyproject.org/

Part III

Machine Learning Techniques applied to IoT

Chapter 7

Fabulos: a Domotic Assistant Agent for Interaction by means of Natural Language

7.1 Introduction

In the last decade, computing environments have become quite pervasive with respect to past. While they have been widely used in professional and working environments, they are now part of our everyday life even in our homes. Household appliances exploit CPU-based equipment since a long time; however, the recent trend is towards *IoT-based Smart Homes* [147], where computing pervasiveness is pushed at high levels: as an example, many modern household appliances also embed wireless communication capabilities; traditional sockets can be easily replaced by *smart sockets* [148] able to measure the power and with the ability of being remotely controlled; smart cameras connected to smart phones can now be easily installed in homes for surveillance applications.

This kind of pervasiveness, while helping our activities, surely presents some drawbacks: in general, each device has its own control App, often provided for a smartphone platform, and mostly these devices/Apps do not interact each other in an integrated manner (even if most of them use standard protocols such as https); in other words, the users is forced to switch among different Apps in order to control and access all the home devices. A fair solution to this issue is to develop an integrated ad-hoc App implementing all the protocols for the devices to be accessed, but such a monolithic application poses other problems in terms of flexibility, updating,

and maintenance when new devices have to be added or changed.

On the contrary, a different approach implies to transfer the integration abilities to another kind of system, rather than the smartphone, but using the latter to perform the interaction: indeed we may think to a sort of "robot", which is able to talk, on one side, with all the devices of the home environment and, on the other side, with the user: in other words, a sort of *personal agent* that virtually lives always at home, acting on behalf of ourselves. Therefore, since such a robot can be seen as a virtual being, one may think to it as able to interact with real people (i.e. the home's householders) using the same *social media* we use everyday, such as Whatsapp, Telegram, Facebook, Instagram, etc.

In such a context, in this work we have developed a software prototype of a *social assistant*, called FABULOS, able to let a user interact with her/his home automation environment via classical social channels. It is composed of three main parts: *(i)* the *Smart Environment Interface*, which includes all the components to interact with home devices; *(ii)* the *Social Network Interface*, that allows access to social media; *(iii)* a *BDI Inference Engine*, i.e. a rule-based AI system that implements the logic of the interaction with the user and the needed automation tasks; and *(iv)* the *Translation Services*, a component able to parse and interpret the sentences provided by the user.

This work, while describing the overall software architecture, focuses on this last component, i.e. the *Translation Services* and, in particular, on the modules to perform the Natural Language Processing (NLP) and extract the real *intentions* associated to the sentences uttered by the user. The algorithms and techniques employed are described in the chapter, as well as a case-study that shows an example of how the proposed approach behaves during a real interaction session.

7.2 Related works

Among vocal assistants, we cannot ignore the two most popular ones, which are Google Assistant and Amazon Alexa. The latter has a wider set of abilities, the so-called *skills*, which are concretely implemented as Apps running in your smart speaker and that let you to do more with your device than what it is capable of out of the box. On the contrary, Google Assistant is better linked with all kinds of

information sources, thanks with its direct connection with the Google search engine. Both accept only a single domotic command in natural language, like *"turn on the air cooler"*, not given in a pipeline, and without making use of any specific conditionals; commands like *"turn on the air cooler only if the room temperature is 25 or higher"* are not properly recognised by these systems, unless making use of additional third part software like IFTTT[149] or others; however this additional software needs communication in cloud, increasing latency times and the complexity of the overall application. Moreover, these systems require the intervention of the user who has to configure specific "applets" that manage the association of commands to devices.

Another platform with higher understanding capabilities, made by the Apple Siri team, is *Viv*. Two years ago, Viv was acquired by Samsung, which announced their AI assistant: *Bixby* will begin to incorporate Viv Labs technology for third-party integration in the market of smartphones, but, so far, its employment in domotic contexts has not considered nor announced.

Another platform worth to be mentioned is Wit.ai[1]. It is an open NLP platform, that can be used via Web or as cloud service, and allows developers to build bot/-conversational applications and devices that can receive and send messages. Wit.ai provides an interface and an API to perform training of human conversations; the objective is to have a platform able to parse incoming messages (voice or text) into a structured data. The process is based on *intents* and *entities*: an intent is simply what the user intends to do (e.g. changeTemperature), while entities are variables that contain details of the user's task. Wit.ai comes with default entity types, such as location, number, and let developers to create their own. Actually, Wit.ai is free (for commercial use too), nevertheless, Wit.ai does not support third-party integration.

Apart the cited commercial products, we can say that, although there is a good amount of literature about Natural Language Understanding (NLU) on IoT environments [150], the join of them inherits much from NLU in robotics. The (common) aim remains, either in IoT or robotics, a set of operations for any available device, each of which is extracted from a string expressed in natural language but with a precise and pre-fixed form. In the light of this, the authors of [151] developed a natural language interface for human-robot interaction which implements reasoning

[1]https://wit.ai/

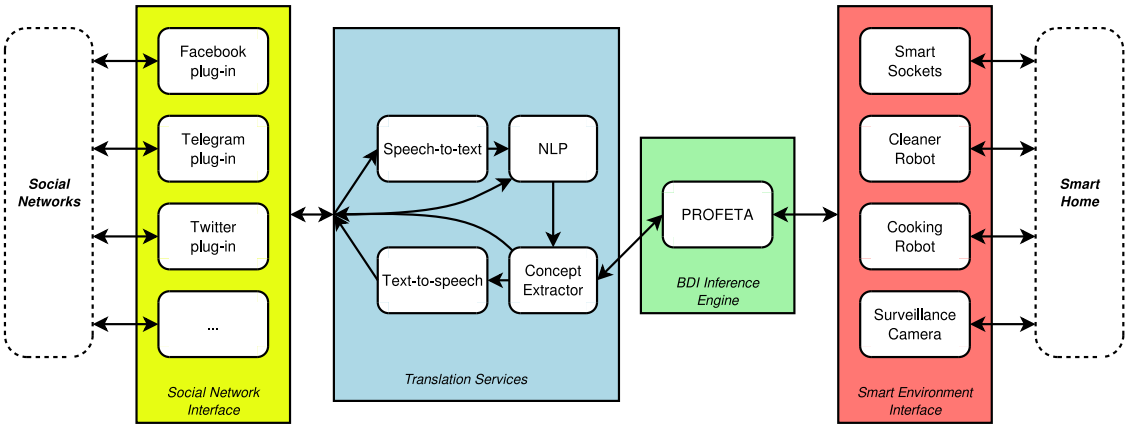


Figure 7.1: The Software Architecture of FABULOS

about deep semantics in natural language, employing methods derived from cognitive linguistics. The system has a complex and powerful interface, but does not face properly the issue of speech recognition, and does not go deep to the general meaning of the words, in order to accept commands in different ways without changing behaviour's rules or adding new ones.

The authors of [152] pursue an information theoretic approach in order to minimise uncertainty in language, by means of apposite clarification questions. The system is based on learning but does not deal with conditionals clauses for subordinating main intentions, neither negations.

The authors of [153] and [154] represent meaning as an ontologically richly sorted, relational structure, based on Combinatory Categorical Grammar (CCG) [155]. In our work, instead of logical approaches, we preferred the dependency parsing of the input sentence, because it's handier for information extraction, plus, the Wordnet features for meaning coding/representation.

In [156], the authors developed a natural language command interpreter called NLCI, which accepts action commands in English and translates them into executable code. Although not used in domotic context, NLCI has in common with our work the usage of a dependency parser for the semantic roles extraction.

7.3 Software Architecture

In this Section, we give an overview of the software architecture of FABULOS by highlighting the various components, their role and their interactions.

Figure 7.1 shows the structure of FABULOS which is basically made of four components: the *Social Network Interface*, the *Translation Services*, the *BDI Engine* and *Smart Environment Interface*.

The *Social Network Interface* has the task of making FABULOS interact with the user. The objective is to let a user "talk" with FABULOS by means of the nowadays widely used social channels, like Facebook or Twitter, and messaging systems, such as Telegram, Whatsapp or Messenger. The user can contact FABULOS using chat or voice messages and be notified of home events through chat/voice messages as well, or even proper posts or tweets. The component runs various "plug-ins", one for each social channel to be interfaced; each plug-in, on one hand, implements the communication protocols for the specific social network and, on the other hand, exploits the API of the Translation Service to notify user requests and get responses from the agent.

Conceptually the Social Network Interface is a simple "protocol translator" and does not implement any form of intelligence or other complex behaviour. On the contrary, the *Translation Service* is one of the most important (and most complex) component of the architecture, since it has the objective of *understanding* and *interpreting* the *real meaning* of messages provided by the user. Since user requests may come either by text or by voice, this component includes the proper modules to perform speech-to-text and text-to-speech tasks, which are in particular implemented by exploiting ready-to-user cloud services[2]. Once that command is in plain text, it must be analysed in order to extract the meaning and thus understand the real intention of the user that uttered the sentence; This task is performed by the components *NLP* and *Concept Extractor*: the former implements syntactic and semantic analysis of the text while the extraction of the user intention, together with possible conditionals is handled by the latter component.

The result of the Translation Services module is a set of *beliefs* that are able to represent the user intention with the associated parameters and conditionals,

[2]In this sense, we made some tests using both Google TTS/STT API and IBM Watson Services; in both cases, we obtained reasonable results.

if present; such beliefs are then processed by the PROFETA engine [157], a BDI rule-based system that is able to trigger the rules representing the computations to be done following such a belief assertion. The aim of such rules is to perform an interpretation of the intention and contact the relevant IoT device to concretely execute the command. This is performed by exploiting the API provided by the *Smart Environment Interface*, a module that includes the component for the interaction with the IoT devices present in the environment; to this aim, each IoT device is represented by a corresponding *driver* that implements all the needed protocols to handle the device itself.

7.4 Extracting Intentions from Utterances

The most complex task of any vocal/text assistant is to extract the real meaning from an utterance provided by the user. This implies to basically perform the following steps:

1. if the sentence has been given in a spoken form, the speech must be converted into text;
2. then, the sentence has to be syntactically analysed, in order to classify semantic terms and determine the relationships between them;
3. finally, the context and the meaning of the semantic terms must be inferred in order to understand user intention.

The first step can be dealt with by means of *speech-to-text* tools; in this sense, many cloud services already exist, including for example Google STT API and IBM Watson; they perform really well, also thanks to the exploitation of Web search that helps to solve ambiguities by comparing all the possible identified sentences with the most recurring ones found in the WWW.

The second step entails the usage of a so-called *dependency parser*, a tool able to analyse the syntax of the text sentence, extract the terms and their relationship in the phrase. Also for this task, many tools exist, such as spaCy [158] or CoreNLP [159], that can be directly used and well serve for the purpose; but the results of a dependency parser cannot be used "as is" and a further context analysis must be applied, which is indeed the third step and the core of our contribution.

Without losing generality, we can assume that the output of dependency parser is a set of *predicates*, in the form $P(t_1, t_2)$, where P is the predicate (relationship) name, and t_1 and t_2 are the terms in that relationship.

In domotic contexts, the sentence will be always in the form of imperative verbal phrase, such as: *Turn off the light in the living room* or *Activate the camera in the garden*. As an example, the first sentence will provide the predicates expressed in the table following table, together with the relevant meaning[3]:

ROOT(ROOT, Turn)	Root term
prt(Turn, off)	Particle
det(light, the)	Determinative (definite) term
dobj(Turn, light)	Direct object
prep(Turn, in)	Preposition
det(room, the)	Determinative (definite) term
compound(room, living)	Compound term
pobj(in, room)	Preposition object

A similar predicate set is generated for the second sentence. In these cases (i.e. imperative verbal form), a dependency parsing will always generate *dobj* (direct object) relations between verbs and objects related to them, which is the main key to understand the intentions inside a command. Still, the dependency parser will also generate *pobj* (preposition object) relations, which give additional meaning about the so-called modificators of the main intentions.

From the set of predicates, we remove those representing articles and create two lists:

- *Intentions*, I, that includes the relations between verbs and objects; and
- *Modificators*, M, that includes relations between verbs and parameters.

We represent such lists as:

$$\begin{aligned} I &= [\ (vb_1, obj_1), (vb_2, obj_2), ..., (vb_n, obj_n)\ ] \\ M &= [\ (vb_{h_1}, mod_1), (vb_{h_2}, mod_2), ..., (vb_{h_m}, mod_m)\ ] \end{aligned}$$

[3]In the following, we will use the predicate names provided by the spaCy parser, which is the tool we employed in our prototype implementation.

$$\text{with} \quad 1 \leq h_i \leq n, i = 1, 2, ..., m$$

Considering the previous example, I and M will be[4]:

$$I = [\ (\text{Turn off, light})\ ]$$
$$M = [\ (\text{Turn off, living room})\ ]$$

In the case we want to include further modificators, such as *Turn off the light in the living room at 12.00*, an additional relation *pobj(at, 12.00)* will be generated. So, taking into account of the occurrence order in a line-to-line reading of the dependency parsing, the list M will be as follows:

$$M = [\ (\text{Turn off, living room}), (\text{Turn off, 12.00})\ ]$$

The dependency parsing is also able to extract *temporal modificators*, such as "tomorrow", "today", "sunday", "monday" (if they are present), that are indeed very relevant for domotic commands. In this case, the dependency parser will generate the additional predicate *npadvmod*[5], that creates a relationship between the verb and the temporal adverb. Temporal modificators are thus collected in another list, T, formed as:

$$\begin{aligned} T \quad = \quad & [\ (vb_{k_1}, tmod_{k_1}), (vb_{k_2}, tmod_{k_2}), ..., (vb_{k_r}, tmod_{k_r})\ ] \\ & with\ 1 \leq k_i \leq n, i = 1, 2, ..., r \end{aligned}$$

If we include, in the sentence above, the term "tomorrow", list T will be as:

$$T = [\ (\text{Turn off, tomorrow})\ ]$$

A further option that is often used in imperative commands is to subordinate the intention to another clause, expressing one or more arbitrary conditions, that employ adverbial clauses like *while*, *when* or the preposition *if*. All these semantic elements will be marked as *mark* by the dependency parser, a predicate that is used by our analyser to build another list called C, that includes the list of couples

[4]Please note that, in constructing the verb, the particle is considered as a part of it.
[5]Noun Phrase Adverbial Modificator

$(entity, property)$, where *entity* is the subject of the condition and *property* is the parameter:

$$C \;=\; [\ (ent_1, prop_1), (ent_2, prop_2), ..., (ent_s, prop_s)\ ]$$

As an example, if the sentence includes the conditional *"... if the temperature is under 25 and the floor is dirty"*, list C will result as follows:

$$C = [\ \text{(temperature, under 25), (floor, dirty)}\ ]$$

After having constructed the lists I, M, T and C, the next step is to let a developer create an inference system through a set of production rules, which takes into account the *meaning* of words, without making use only of a mere strings-matching of parts of domotic command. To this aim, let us define the *Domotic Vocabulary* as a set of all the *verbs* and *objects* that can appear in a domotic context. If we consider several languages, we have $V_D \subset \bigcup_{n=1}^{N} V_i$, where every V_i is a distinct vocabulary of a given language (for instance $V_D \subset \{V_{EN} \cup\ V_{IT} \cup V_{FR}\}$). Starting from V_D, we consider the following subsets:

$$\begin{aligned} Verbs_D &= \{v \in V_D \mid v\ is\ a\ verb\} \\ Objects_D &= \{o \in V_D \mid o\ is\ a\ noun\} \end{aligned}$$

Let F_M be the *Meaning function* as:

$$F_M : V_D \rightarrow C_D$$

$C_D \subset S$, with S = the space of all possible strings.

F_M is a special function, which encodes and maps together, with the same value, all terms of V_D that share the same meaning, in the domotic context, in whatever language. As an example, the terms: "air conditioner", "conditioner", "air cooling", "cooler" and "climatization", refer to the same equipment and thus they will mapped, through F_M, to a same string. In the same way, also verbs are mapped through F_M in such a way as to have the same results for synonyms.

The Meaning Function is used as follows: first the *Total Intentions* I_T set is formed as all pairs (v, o) from $Verbs_D$ and $Objects_D$:

$$I_T = \{\ (v, o) \mid v \in Verbs_D, o \in Objects_D\ \}$$

We say that $I_A \subset I_T$ is a set of *acceptable intentions*, for a given vocabulary $V_x \subset V_D$, when exists a program P which implements $P_{V_x}(F_M(v), F_M(o))$; in other words:

$$I_A = \{\ (v, o) \in I_T \mid \exists P_{V_x}(F_M(v), F_M(o))\ \}$$

As the program $P_{V_x}(F_M(v), F_M(o))$, we consider a clause, in a rule-production system, that is able to match the results of $F_M(v)$ and $F_M(o)$, and execute the relevant code to concretely perform the action. In FABULOS, this is performed by using PROFETA and, in particular, by asserting the belief `Intent` whose parameters will be:

- the verb v;
- the object o;
- the set of modificators M;
- the set of temporal modificators T;
- the set of conditionals C.

When a sentence is analysed, an `Intent` belief is asserted for each possible meaning provided by the Meaning Function, but it's up to the developer to write the rule corresponding to the specific meaning she/he intend to catch. This aspect will be clarified in the next section, where a concrete example will be shown.

The algorithm that handles sets I, M, T and C and produces the `Intent` belief, exploiting the Meaning Function, is expressed in the Algorithm 2 and constitutes the last step of the meaning extraction process.

7.5 Case-Study

In this Section we report a simple case-study that shows how the extracted intention can be exploited to program some inference rules executing what the user said.

Algorithm 2

procedure FINDINTENTS(I, M, T, C)
  for $i \leftarrow 1, n$ **do**
    $m_{v_i} \leftarrow F_M(v_i)$
    for $j \leftarrow 1, n$ **do**
      $m_{o_j} \leftarrow F_M(o_j)$
      if $C = \emptyset$ **then**
        assert_belief(Intent(m_{v_i}, m_{o_j}, M, T))
      else
        assert_belief(Intent(m_{v_i}, m_{o_j}, M, T, C))
      end if
    end for
  end for
end procedure

In our prototype implementation of FABULOS, since the inference engine used is PROFETA, which is Python-based, also all the modules of FABULOS are implemented in Python. Similarly, for what concerns the dependency parser, we exploited the spaCy [158] framework, which is also in Python, (although we made also some tests with CoreNLP Stanford [159]) due to a wider range of supported languages.

As for the Meaning Function, we considered the convenient biunique correspondence between WordNet [160] synsets and general meanings. On this basis, to create a rule which expresses the meaning of a single intention, we have to choose the correct synset among all available ones which are the same thorough different languages, and then use it into a production rule; this must be done for both verbs and objects. For instance, to represent the action of changing the air conditioner setting, the synsets *specify.v.02* and *air_conditioner.n.01*, among all synsets encoded in WordNet, express exactly what we mean. So, replacing meanings with synsets, the production rule which expresses our intention became as in Algorithm 3. Please consider that, for the sake of brevity, we omitted here the details about PROFETA syntax that is indeed reported in [157], but we can clarify that the rule shown means "when the `Intent` belief is asserted, with those parameters, execute the action `set_air_conditioner`".

Now, considering also the objective to handle the English language, the usage of synsets within the production rule will let PROFETA to accept all the beliefs containing anyone of those lemmas within the synsets, as the sentence (restricted

Algorithm 3

```
+Intent("specify.v.02",
        "air_conditioner.n.01",
        M,T) >> set_air_conditioner(M,T)
```

only to this case) were one of the following:

Set the air conditioner at...
or
Fix the air conditioner at...
or
Set the cooler at...
...
etc.

Now let us suppose we give FABULOS the following command:

> *Set the air conditioner at 26 today at 12.00 and turn off the light tomorrow at noon if the temperature is under 25*

The sets I, M, T and C derived by the technique described in Section 7.4 will be:

$$\begin{aligned} I &= [(\text{set}, \text{air\_conditioner}), (\text{turn\_off}, \text{light})] \\ M &= [(\text{set}, 26), (\text{set}, 12.00), (\text{turn\_off}, \text{noon})] \\ T &= [(\text{set}, \text{today}), (\text{turn\_off}, \text{tomorrow})] \\ C &= [(\text{temperature}, '\text{under25}')] \end{aligned}$$

On this basis, once the sets above have been obtained, the FINDINTENTS procedure will assert the following beliefs:

Intent("put.v.01","air_conditioner.n.01",M,T,C)
Intent("determine.v.03","air_conditioner.n.01",M,T,C))

Intent("set.v.04","air_conditioner.n.01",M,T,C))
Intent("specify.v.02","air_conditioner.n.01",M,T,C))
....
....
Intent("dress.v.16","air_conditioner.n.01",M,T,C))

Indeed, according to WordNet, the verb *set* is contained as a lemma in 25 synsets, such as: "put.v.01", "determine.v.03", "specify.v.02", "set.v.04", etc., but it is the "specify.v.02" which has the meaning we wanted to express by means of the production rule. The noun *air_conditioner* is contained as lemma in only one synset, which is "air_conditioner.n.01". Therefore, only the fourth Intent will be recognised by the program in Algorithm 3, thus executing the relevant action; all the other beliefs will be ignored since they are not related to any matching rule.

7.6 Conclusions

In this work, we presented a software architecture, based on the BDI framework PROFETA, for the implementations of software agents, which allow users to interact with their domotic devices via most commonly used social network channels. Although the infrastructure uses social media as data input, it has the merit of not being entirely dependent on cloud services, thus one can easily integrate it with local Speech-To-Text engines like Sphinx[161] or others, letting FABULOS work also in absence of the internet connection; this would make the system more reliable and preferable than the well-known cloud-dependent domotic systems.

The proposed architecture, is capable of processing a domotic command given in many different shapes, extracting the very meaning of all its terms. In addition, our Natural Language Processor works regardless the text message, because it does not compare the input with prepared texts, but analyses in depth the grammar structure of the sentence. In this way, we can parse the sentence and understand its meaning (the intentions and the entities) without regard the language and also working with synonyms. Futhermore, our software solution is able to manage a flow of intentions with conditionals and preserves privacy users processing all data locally.

As future work, we aim at testing and integrating the proposed software solution into a hardware which acts as hub for IoT home devices.

Chapter 8

A Neural Network Model for the Solar Module Power Prediction

8.1 Introduction

Renewable energy has always aroused great interest in the field of energy production, not only for industrial use but, in recent times, also for private and city usage. Among renewable energy, photovoltaics is undoubtedly one of the most promising solutions and, for this reason, the production of solar cells has gained a lot of attention among companies producing semiconductor devices.

Typically a solar module is made up of a set of 60 to 72 solar cells, according to the output power that is needed and assembled together through a soldering process. One key aspect of a solar module, and hence of a cell, is its *efficiency*, i.e. the transfer factor from solar irradiance to power output, that should be always kept as maximum as possible. Obviously the efficiency of a solar module depends on the efficiency of each cell composing it, but unfortunately such a dependence cannot be easily understood due to a large number of factors and parameters that are related to the characteristics of each cell.

Indeed, cell production is based on many steps that, starting from a silicon wafer through a series of chemical, mechanical and electrical processes, finally completes a cell, and such steps are in general performed repetitively by automated machines. However, despite the automation of the whole process, due to many factors like wafer quality, chemical reactions, etc., the produced solar cells may have different characteristics from each other, and thus each feature a different efficiency factor. The bad news is that once a PV panel is made of cells with many different efficiency

factors, the result may be a product with a very low overall power production capability.

The practical way to deal with such a problem is to perform (in the automated production line) an irradiation test aimed at determining the efficiency factor of the cell and then to classify the cell according to specific ranges of the factor itself. On this basis, making a PV panel by using cells of the same class should imply, at first sight, a final product with a power factor that is close to that of the composing cells. However, tests made on real products prove that this statement is often false. Indeed, since the aim is always to produce PV panels with power factors as high as possible, an ideal way to deal with such a problem is having the possibility of *predicting* the final efficiency of a panel starting from a set of given cells.

In this context, this work proposes a neural network model capable of predicting the maximum final power of a PV module considering as input a set of real solar cells which would be used to assemble the module itself. The network is based on a large set of electrical parameters that completely characterise each cell composing the module. In this way, the production line can simulate a final solar module, without actually having to assemble it, with the objective of understanding whether the final performances would be acceptable. Also, the proposed model works with more than one class of solar cells, allowing experiments by combining different types of solar cells. In this way, the solar cell classification will be simplified, decreasing the number of solar cell classes and improving the performance of the production line. Moreover, the same model was trained and tested using realistic datasets supplied directly from the production lines, improving and increasing the degree of reliability with the possibility of being used in the production line. Finally, the same model can be used with the help of an ad-hoc desktop/server application to make predictions also through a private network, using a set of RESTful APIs provided by the software itself through HTTP protocol.

The proposed work briefly introduces the fundamental characteristics of solar cells and modules, highlighting, through example, the problem addressed. Soon after, the dataset used for the training and the analysis of the latter will be introduced, considering the correlations between the various inputs and the possibility of being able to reduce the number to train the neural model. Therefore the structure of the proposed neural network model, the technologies and algorithms used, and the

definition of error according to the model itself, will be introduced, presenting the first results of the same model. Also, an improvement to the same model will also be introduced allowing it to be used with more than one class of solar cells, permitting experiments about the combination of near solar cells classes. Finally, the latest test results of the model, the application, and technologies used will be presented, with which you can directly use the model in production or for experimentation.

8.2 Related Works

In the field of IoT and deep learning technologies, the application and use of these in an industrial production environment are becoming an important requirement for enterprises which are changing their manufacturing architectures. Although this chapter is focused on the training and use of a neural network to predict the efficiency of the final product, a few related works are similarly aimed at solar energy production[162, 163, 164, 165, 166, 167].

Among them, in [162] a solar cell status classification is presently based on deep learning and operated using electroluminescence (EL) images. The basic idea is to detect and classify the solar cells which suffer from some production or lifetime defect. The proposed neural network model seems to be useful in terms of waste of hours and manpower. The default analysis process of the EL images is a manual job and the amount of these images is about 90 million per solar energy production field. Therefore, the authors propose a convolutional neural network based on images analysis which classifies the solar cells in a set of two classes, either good or defect. A similar approach was introduced in [163], where authors proposed a convolutional neural network model which use low-resolution satellite photos as input.

The authors of [164] presented a neural network capable of detecting power losses to be attributed to the position of the sun. The three angles (altitude, azimuth and incidence solar angles) were used as the input dataset and represent the position of the sun in the sky. The authors used a fully connected neural network with two hidden layers and only one output which is the power production prediction relative to the position of the sun. The model results useful to detect the best or wrong placement position of the solar panel, increasing the final efficiency of it. Moreover, the proposed model can operate independently from the geographical

location of the solar panel. In [165], the authors present a similar result. This paper proposes an application of a neural network that can predict the maximum output power of a solar panel array under the effects of some non-uniform shadows, dependent on its geolocalisation. The work is based on the difficulty of calculating and estimating the shading factor (which is the ratio of shaded/non-shaded area) on the solar panel. The hardest part of this operation arises with the dynamic shape and position of the shadows. The approach consists in the estimation of the shadow ratio, which is relative to the altitude and angles of the position of the sun in the sky. Subsequently, this information, with the irradiation levels, is used to train the neural network model, obtaining the maximum output power as output.

The authors of [166] introduced methods to forecast solar power generation in a solar energy plant. The methods take into account some new variables of the environment such as wind, cloud coverage, temperature, rainfall and humidity as well as sun altitude and sky position angles. The neural network model proposed was obtained and compared using a branch of regression functions. The training dataset is based on real data over one year of operational solar panels and consists of seven inputs and one output that is the predicted power production. Also, the output will be weighed with the degradation time of solar panels.

Lastly, a neural network model for the prediction of the maximum power point of a solar panel was presented in [167]. More specifically, the authors used a combination of back-propagation and radial basis networks (RBF) to understand which one of these two networks could be the best one. The training dataset was generated virtually (considering the I-V solar cell characteristic) and consists of a set of two inputs (the irradiation and temperature of the solar cell) and the maximum power point as output. The results confirm that the RBF network was the best choice for the authors' work.

8.3 Automated Assembly of Solar Modules

Basically, a solar module, also called solar panels, is a single photovoltaic panel that is an assembly of interconnected solar cells, enclosed in a weatherproof package. The solar cells absorb sunlight as a source of energy to generate electricity. The solar modules will be merged into an array of modules used to supply power to buildings.

A solar array of modules consisting of higher-energy producing solar modules will, therefore, produce more electricity in less space than an array of lower-producing modules.

The solar cells' efficiency and output power can vary depending on the type and quality of solar cells used. A solar module can range in energy production from 100-380 Watts of direct current (DC) electricity. Solar cells are made using silicon crystalline wafers which are similar to the wafers used to make computer processors. The silicon wafers can be either polycrystalline or monocrystalline and are produced using several different manufacturing methods. The most efficient type is monocrystalline which is manufactured using a method known as Czochralski process[168].

Solar modules are manufactured in an automated assembly line which is a set of machines and robots that operate automatically in a precise way. These machines communicate among themselves using a specific application network protocol[169] which makes the synchronisation according to its production and status. Typically, these machines send their information to a main server that notify the production status and other important information, such as production faults[170]. Starting with individual cells, the machines take cells and process them into "ready to mount" modules. Several machines contribute to the manufacturing process. We can break down this manufacturing process into two sections: the Solar Cell Production and the Solar Cell Assembly, that is the Solar Module Production.

In the Solar Cell Production chain the solar cells are created and classified according to their electrical performance[171], which are tested under simulated sunlight. When the Solar Cells are made, tested and classified, these are stored in specific bins, each one for each different efficiency class.

A solar cell is basically a semiconductor device[172] that converts the energy emitted by the sun into electric power, as the product of electric current (I) and electric potential (U).

$$P_{(W)} = I_{(A)} \cdot U_{(V)} \tag{8.1}$$

Solar cells produce photocurrent while light strikes on it, while the photovoltage or potential difference is independent of the intensity of incident light, whereas the current capacity of a cell is proportional to the intensity of incident light as

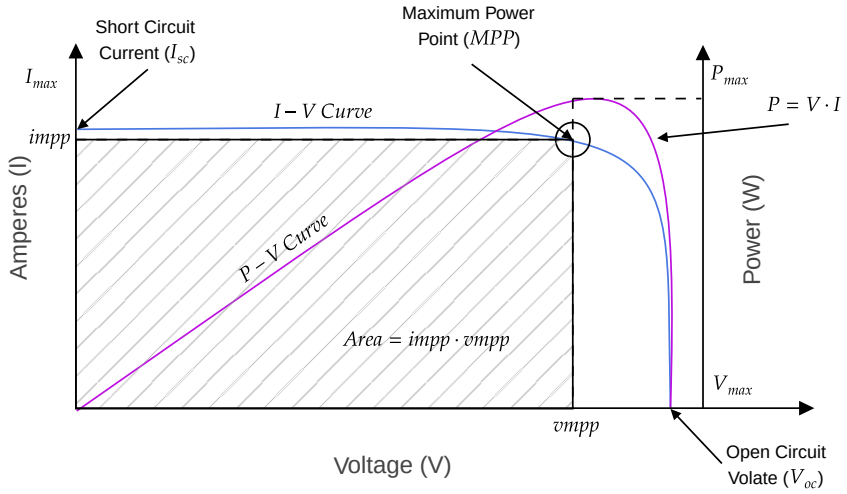


Figure 8.1: Solar Cell $I - V$ Characteristic

well as the area that is exposed to the light. A combination of photovoltage and photocurrent values at which a solar cell can be operated is called a working point. These values at short and open circuit conditions are called short-circuit current (I_{sc}) and open-circuit voltage (V_{oc}).

The I_{sc} is the maximum current that a solar cell can deliver without harming its own constriction. It is measured by short circuiting the terminals of the cell at the best optimised condition for producing maximum output. As the current production also depends upon the surface area of the cell exposed to light, it is better to express maximum current density instead of maximum current. The maximum current density is the ratio of I_{sc} to exposed to the surface area of the cell.

$$J_{sc} = \frac{I_{sc}}{A} \tag{8.2}$$

where J_{sc} is the maximum current density, I_{sc} is the short circuit current and A is the area of the solar cell.

The V_{oc} is the measured voltage of the solar cell when there is no load connected to the solar cell. This value depends on the manufacturing quality, modes and temperatures, but not depends on the intensity of light and its area of coverage.

The main electrical characteristics of a solar cell (and solar modules) are summarised in the relationship between the current and voltage produced on a solar cell

I-V characteristic curve (fig. 8.1).

A current–voltage characteristic (called I-V characteristic) of a solar cell is a plot of all possible working points in a considered range. I-V characteristic provides the information required to configure a solar system so that it can operate as close to its optimal peak power point (MPP) as possible. There is one combination of current and voltage where the power generated by the solar cell reaches its maximum (I_{mpp} and V_{mpp})[173]. This point on the I-V characteristic of an illuminated solar cell is called the maximum power point (P_{mpp}).

$$P_{mpp} = I_{mpp} \cdot V_{mpp} \tag{8.3}$$

There is an additional parameter for the characterisation of the Solar Cell that is the Fill Factor (FF). It describes the amount with which the $I_{sc} - V_{oc}$ "rectangle" is filled by the $I_{mpp} - V_{mpp}$ "rectangle" in the I-V characteristic plot. The Fill Factor value gives an idea of the quality of the array and the closer the fill factor is to 1, the more power the array can provide.

$$FF = \frac{I_{mpp} \cdot V_{mpp}}{I_{sc} \cdot V_{oc}} \tag{8.4}$$

Finally, the efficiency η of a solar cell is the ratio between the maximum electrical power that the solar cell can produce compared to the amount of the solar irradiance hitting the cell.

$$\eta = FF \cdot \frac{I_{sc} \cdot V_{oc}}{P_{sun}} \tag{8.5}$$

Therefore, in an automated factory of solar cells, the classification of these is based on the P_{mpp} value of the single solar cell. In this way, the production of the final solar module can be determined by what kind of a solar module is desired, in terms of the maximum power point of the solar module that is wanted.

Taking into account all of the above formulas, and in particular taking into consideration the formula in 8.3 that is, a group of Solar Cells with the same classification and therefore with similar characteristics. For instance, lets suppose that the maximum power point of a single solar cell is:

$$\boldsymbol{P_{mpp}} = I_{mpp} \cdot V_{mpp} = \mathbf{5.5}\boldsymbol{W} \tag{8.6}$$

An industrial solar module, typically, is composed by 60-72 solar cells and these will be assembled in a connection series. Supposing we have 72 cells, then the final maximum power point of the solar module will be:

$$\boldsymbol{P_{mpp_{(module)}}} = P_{mpp} \cdot 72 = \mathbf{396}\boldsymbol{W} \tag{8.7}$$

In reality, the final power of the module will be lower than the mathematical result because it is affected by factors strictly related to the module processing steps, such as interconnection, stringing and lamination that lead to optical gains and resistive losses, as well as factors related to the used glass materials that determine undesired absorption of light. These phenomena are described by the cell-to-module power ratio (CTM_{power})[174] that describes the ratio of the power output of the solar module to the sum of the power of the cells embedded in the module. This metric quantifies the power loss equal to 1-2% of the ideal power output given by the formula 8.3.

A proposed solution is the mathematical formalisation of a predictive model which can predict, given a set of solar cells, a possible P_{mpp} of the solar module in output, using the same classification of solar cells of more than one of these. The model will be used to predict and get the P_{mpp} of solar modules without any costs, to experiment new ways of classification and to demonstrate that power loss is not related to the use of different classes of solar cells. Moreover, the model will be used to experiment and improve the production of a new set of solar cell recipes. The final model will be based, obviously, on the actual state of the manufacturing environment and production, but can be adapted to any production environment.

8.4 Dataset Analysis

In order to implement the model, the process data used was generated during engineering tests in the Solar Module Production Line. The first dataset was created considering 2490 modules built up with cells belonging to the same class (BIN). For each module the input of the model is given by means of the electrical parameters of the cells stringed inside the module since the standard deviation of the statistical distribution of the cells belonging to a specific class is very low. Then we can approximate the raw data to the mean values. The output target is instead given from

	count	mean	std	min	25%	50%	75%	max
Tcell	2490.0	31.101824	0.924260	28.950531	30.514487	30.974315	31.587993	33.647268
Voc	2490.0	0.728333	0.002960	0.715065	0.726716	0.728993	0.730396	0.734098
Vmpp	2490.0	0.616127	0.006148	0.586048	0.612707	0.617391	0.620138	0.628127
Isc	2490.0	9.353728	0.029870	9.145561	9.334788	9.353141	9.371336	9.441513
Impp	2490.0	8.797918	0.058547	8.441078	8.774798	8.809099	8.831255	8.900302
Jsc	2490.0	38.287879	0.122267	37.435783	38.210340	38.285481	38.359947	38.647207
Pmpp cell	2490.0	5.420767	0.081241	5.076663	5.371564	5.436099	5.482212	5.557193
FF	2490.0	79.567487	0.866929	75.100406	79.248990	79.799415	80.080608	81.015388
Eta	2490.0	22.188978	0.332546	20.780444	21.987568	22.251736	22.440497	22.747420
Insol	2490.0	999.649397	0.100482	999.350802	999.577315	999.641897	999.713865	1000.019523
Rser	2490.0	0.005362	0.000493	0.004527	0.005056	0.005250	0.005559	0.009030
Rshunt	2490.0	545.977494	97.227755	5.078913	499.935172	547.900248	599.731721	913.607184
Pmpp Target	2490.0	374.419312	5.539508	350.211700	372.052025	375.366837	378.132530	385.930786

Table 8.1: Original dataset statistics

the final value of Pmpp of the solar module (Pmpp Target). Regarding the solar cells features, the dataset is composed by the following fields: Tcell, Voc, Umpp, Isc, Impp, Jsc, Pmpp cell, FF, Eta, Insol, Rser and Rshunt, which are described in Table 8.2.

A preliminary analysis of the dataset was made. First of all, statistical information was recovered (Table 8.1) and saved to be used subsequently for the normalisation of the dataset. These information will be normalised and used also for the training, test and production environment. Secondly, we analysed the dataset and its correlations among the information. As result, a heatmap of correlations was obtained (Figure 8.2). The verification of correlations is an important part of the exploratory data analysis process[175]. This analysis is used to decide which features affect the target variable the most, and in turn, used in predicting this target variable. In this way, a first reduction of highly correlated features for the model is possible. Some interesting correlations shown as can be seen in the heatmap in Figure 8.2. Among this information some seem to be useless as it might confuse the model. In fact, the Isc and Jsc features seem to be highly correlated, as each one of these can represent the other one, therefore the Isc feature will be used for the training of the model and the Jsc will be dropped. The same thing happens for the Eta and Pmpp_cell features, which in this case it is more convenient to remain on "Pmpp" nomenclature. The other feature that can be dropped for low correlation is the Insol feature. Lastly, as the Table 8.1 and Figure 8.2 shows, there are some very important points made about the series and Shunt resistances (Rs and Rshunt). These resistances have a strange/low correlation with the other inputs and

Inputs description		
Name	**Description**	**Unit of measurement**
Tcell	The test exposition temperature of solar cell	Celsius (C°)
Voc	The open-circuit voltage of the solar cell	Volts (V)
Vmpp	The maximum power point voltage of the solar cell	Volts (V)
Isc	The maximum current of the solar cell	Amperes (A)
Impp	The maximum power point current of the solar cell	Amperes (A)
Jsc	Current density of solar cell	Amperes per square metre (Am^{-2})
Pmpp cell	The maximum power point of solar cell	Watts (W)
FF	The Fill Factor of solar cell	Percentage (%)
Eta	The efficiency of solar cell	Percentage (%)
Insol	The Solar Cell Insolation	Watts per square metre (Wm^{-2})
Rser	The series resistance of solar cell	Ohm (Ω)
Rshunt	The parallel/Shunt resistance of solar cell	Ohm (Ω)
Pmpp Target	The final maximum power point of solar module	Watts (W)

Table 8.2: Original dataset inputs description

the standard deviation of these values is really high (in particular the Rshunt). It must be said, however, that can be a physical consequence of the serial assembly of the solar cells and therefore the high standard deviation is justifiable, but it can also confuse the final neural network model. For this reason the resistances are not considered as inputs for the model. Subsequently, a principal component analysis (PCA) was made.

The PCA is commonly used for dimensionality reduction of the inputs, projecting each information point onto only the first few principal components to obtain lower-dimensional data, while preserving as much of the data's variation as possible. Basically, the PCA generates a new set of features that can describes the original

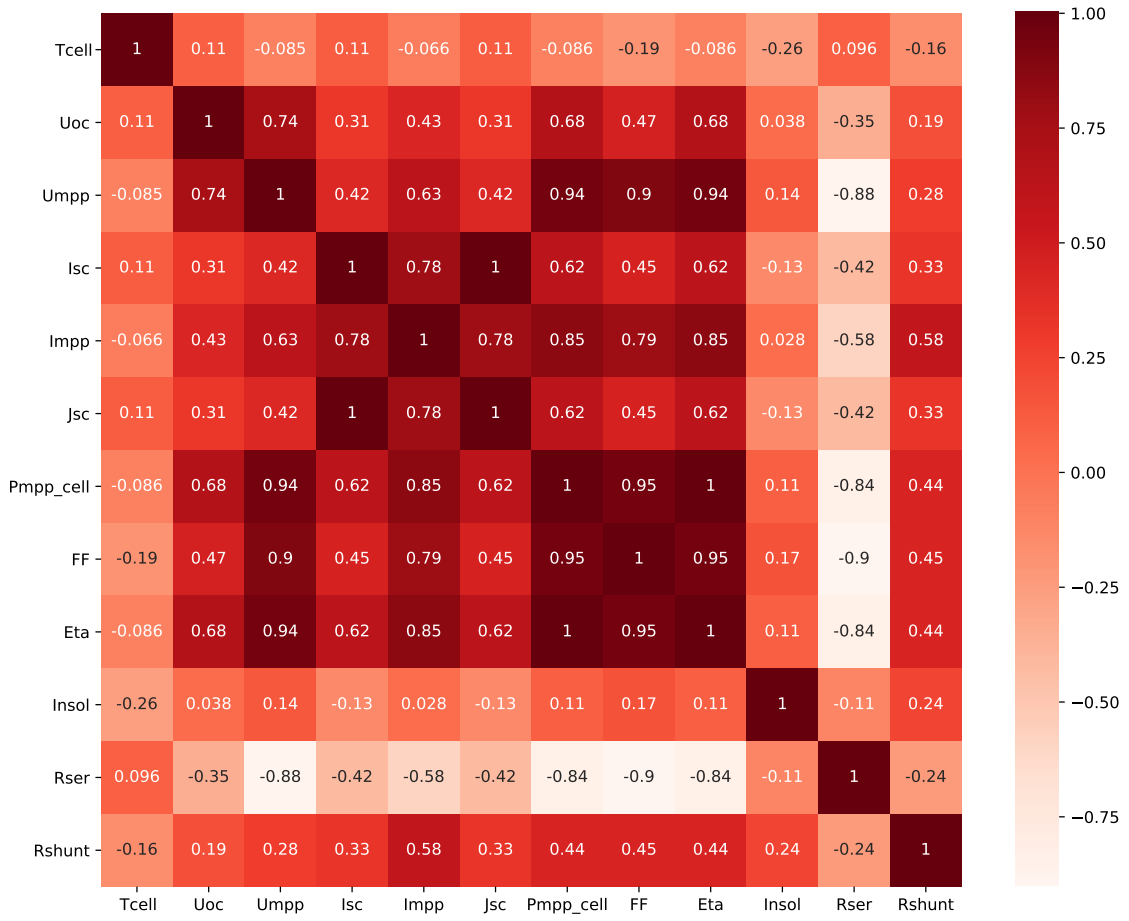


Figure 8.2: Heatmap of the normalised original dataset correlations

dataset, in variance percentage.

Figure 8.3 shows the results of PCA related to our dataset. It shows that about 6 or 7 of the new features generated can describes our dataset, instead of original 11 features. The problem is that the new features cannot be used in a real production environment, but the result of PCA gives some hints and intuition of how many original features can be used to describe the dataset in some total way. Considering the results of PCA and the correlation analysis made before, it is deduced that the features that will be used are 7 and these are Tcell, Voc, Umpp, Isc, Impp, FF and Pmpp Cell (an example of input dataset, not normalised, is shown in Table 8.3).

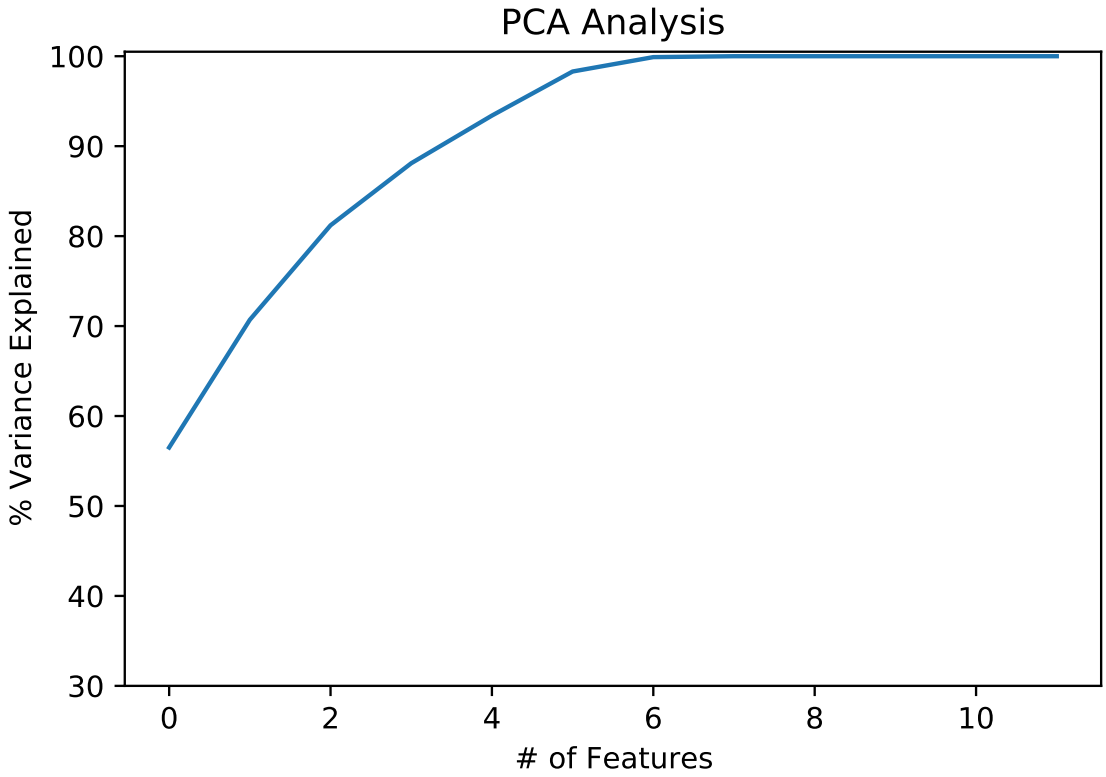


Figure 8.3: Principal component analysis of the normalised original dataset

Tcell	Voc	Umpp	Isc	Impp	FF	Pmp Cell	Pmpp Target
30.82	0.73	0.62	9.41	8.89	80.57	5.56	384.60
...	...	...	...	...	...	...	...

Table 8.3: Example of input dataset (not normalised)

8.5 Model

The training phase of the model and, therefore, the prediction phase, uses the normalised input data, using the original dataset statistics shown in Table 8.1. These statistics are saved and reloaded to be used in the training phase and in every future prediction phase, because the model will be trained with a specific normalised dataset. Then, the domain of the inputs must be the same. For the definition, building and training of the model, Keras[1] technology was used.

[1] https://keras.io/

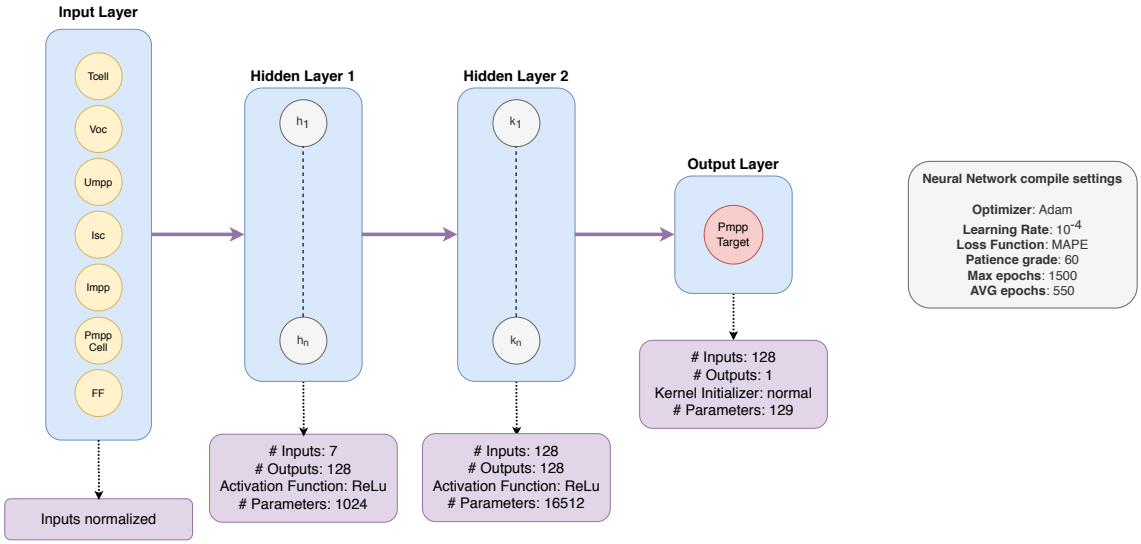


Figure 8.4: Neural Network Model Structure

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow[2]. Keras could be used also with other machine learning engines but TensorFlow (version 2) was chosen.

TensorFlow bundles together a slew of machine learning, deep learning models and algorithms which makes them useful by way of a common metaphor. It also uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++. In fact, the actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together. This is the most important thing because, at the end of model training, TensorFlow permits to convert the model to be used with other technologies, such as TensorFlowJS[3].

As Figure 8.4 shows, the model is composed by 4 layers. The first one is the input layer, where the inputs are passed normalised. The second and third one (the hidden layers) are the same, each one with 128 neurons as input and output, except for the second one that has the same number of inputs as input. The final layer is the output layer, that has obviously only one output, that is the Pmpp Predicted.

[2] https://www.tensorflow.org
[3] https://www.tensorflow.org/js

These values were chosen after some trial and error experiments in order to reach the best performance in terms of network structure, computational complexity and values of loss and accuracy.

The activation functions of two hidden layers is the rectified linear activation unit function (ReLU function defined in 8.8)[176], and is a simple calculation that returns the value provided as input directly, or the value 0 if the input is 0 or less.

$$\boldsymbol{ReLU} = \max(0, x) \tag{8.8}$$

ReLU is linear for all positive values, and zero for all negative values. This means that ReLU is cheap to compute as there is no complicated math. It converges faster, being linear, and is sparsely activated, since ReLU is zero for all negative inputs. It is likely for any given unit to not activate at all. Sparsity results in concise models that often have better predictive power and less overfitting or noise. In a sparse network, it is more likely that neurons are actually processing meaningful aspects of the problem. Finally, a sparse network is faster than a dense network, as there are fewer things to compute. To compile the model, to adjust the weights of the final model, the Adam[177] Optimiser was used.

Adam optimisation is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The Adam optimiser is computationally efficient, has little memory requirement, invariant to diagonal re-scaling of gradients, and is well suited for problems that are large in terms of data and parameters.

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon} \tag{8.9}$$

where w is model weights and η is the learning rate. The learning rate η was set at 10^{-4}.

As loss function the Mean Absolute Percentage Error[178] (MAPE) was used. The MAPE function is a measure of prediction accuracy of a forecasting method in statistics as well as being a loss function for regression problems in machine learning. This has proven to be perfect for this problem.

$$\mathrm{M} = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right| \tag{8.10}$$

Model Compile Settings	
Optimiser	Adam
Learning Rate	10^{-4}
Loss Function	MAPE
AVG epochs	550

Table 8.4: Neural Network Model Settings

where A_t is the actual value and F_t is the forecast value.

Finally, the training of the model was monitored by a callback to early stop. The early stop callback monitors the value of loss with a patience grade of 60. In this way, the training stops before the maximum epochs. For the final model, the training stops at about 550 epochs obtaining scores: loss: 0.388; mean absolute error: 1.4509 and mean squared error: 8.9649.

8.5.1 Error Model Definition

Since the problem being treated is a regression problem, unlike classic classification problems, it is necessary to define what is correct and what is not. To do that, it is necessary to remember that a loss of power of approximately 1-2% is acceptable (e.g. the 1% of $374W$ is $3.75W$) and it is a good choice to define a threshold for the Predicted Pmpp as close as possible to the 1% of the mean of Pmpp Target.

Let be $\Delta\epsilon$ the Euclidean Distance between the **Pmpp Target** and the **Pmpp Predicted**, λ the 1% of the mean Pmpp Target of the training dataset (as the trained model is 3.5) and Θ the closest threshold to λ (for the trained model it is can assumable a value equal to 2.5):

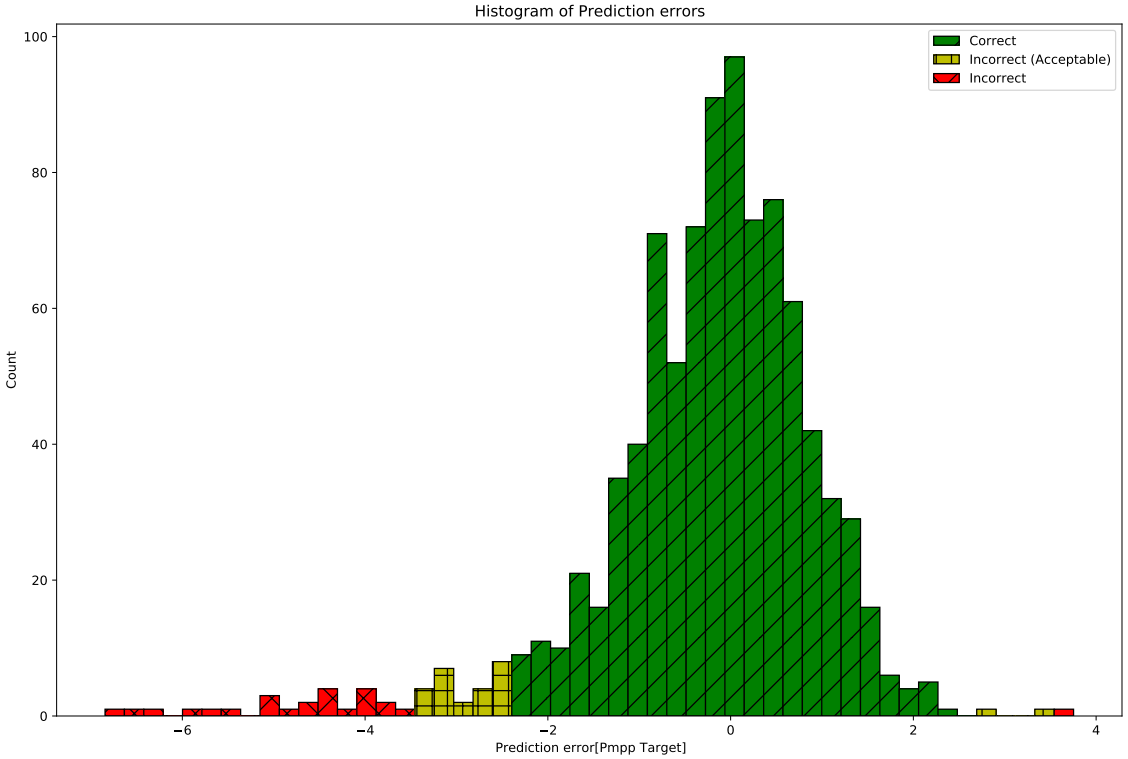


Figure 8.5: Distributions of prediction results (with one BIN)

$$\text{Let } \Delta e = |Pmpp_{(Target)} - Pmpp_{(Predicted)}| \; ; \tag{8.11}$$

$$\lambda = 1\% \text{ of } Pmpp_{(Target)} \text{ dataset average; } (\lambda \simeq 3.5);$$

$$\Theta = \text{closest acceptable value to } \lambda; \; (\Theta = 2.5);$$

$$\text{then:}$$

$$Prediction = \begin{cases} Correct, & \text{if } \Delta\epsilon \in [-\Theta, \Theta] \\ Incorrect \text{ } (but \text{ } acceptable), & \text{if } \Delta\epsilon \in [-\lambda, -\Theta[\cup]\Theta, \lambda] \\ Incorrect, & otherwise \end{cases} \tag{8.12}$$

8.5.2 Achievements and Observations

As test dataset, the dataset used consisted of **922** solar modules. Predictions

	Pmpp Target	**Pmpp Predicted**	**Error**
count	922.000000	922.000000	922.000000
mean	376.116134	376.381714	-0.265582
std	4.259103	3.962520	1.235558
min	358.869812	359.779572	-6.846039
25%	373.611473	373.984764	-0.779381
50%	376.476791	376.845276	-0.101639
75%	379.210892	379.346252	0.481018
max	384.602509	385.177979	3.753174

Table 8.5: Prediction results statistics (with one BIN)

results, as the Figure 8.5 shows, matched **875** modules of 922 given, with **47** incorrect predictions. Therefore, the model achieves an accuracy of prediction about **94.90%** (only the Corrected predictions were considered). Another aspect that gives good results is the statistical information about the errors. As Table 8.5 shows, the standard deviation and the mean of Error are very low. The predictions results demonstrate that the neural network model can predict the final maximum power point of a solar module with high prediction accuracy, and this seems acceptable and can be used in engineering tests to improve the solar cells and solar modules production. Nevertheless, the model can also help to improve the production logistics, such as material transportation, reducing the number of solar cell classes, and using more than one solar cell class, with negligible loss of power. To this purpose, new engineering tests have been made by assembling solar modules with two different "nearby" solar cell classes. The results have been used to improve the model.

8.6 Improved Model

To improve production logistics, the transportation of production materials and the solar cells classification, the reduction of solar cells classes is required. For that reason, a new set of experiments, about the assembly of solar modules with solar cells belonging to contiguous classes were made to verify the possible power loss from a practical and theoretical point of view. The power loss, from a theoretical point of view, is related to the serial connection of solar cells and therefore, the final

Tcell	**Voc**	**Umpp**	**Isc**	**Impp**	**FF**	**Pmp Cell**	**BIN**	**MIN_BIN**	**COUNT_CELLS**	**Pmpp Target**
31.03	0.73	0.62	9.44	8.90	80.61	5.56	17	1	60	385.20
31.99	0.73	0.60	9.40	8.89	80.893	5.58	18	0	12	385.20
...	...	...	...	...	...	...	...	...	...	...

Table 8.6: Example of input dataset with two classes (not normalised)

Pmpp Target	Bin Min	Bin Max	Num Cells Min	Num Cells Max	Pmpp Predicted Min	Pmpp Predicted Max
385.20	17	18	60	12	381.90	384.10
...	...	...	...	...	...	...

Table 8.7: Example of final dataset after predictions with two classes (not normalised)

power of the module settles to the power relative to the class of the solar cells with lesser efficiency.

In this case the obtained model cannot correctly predict the final power of solar module. Therefore, the idea is to use the same model to predict the final power of the solar modules, adding this new information about the final module.

To introduce a new structure of the input dataset, a new way is needed to use the model, as the Table 8.6 shows. In this case, the dataset represents a set of solar modules which are made with two different classes of solar cells.

The dataset contains in rows of two by two, the same solar module and the average of solar cells features, considering their respective classes. In the dataset there was also some information added about the classes such as the relative class of solar cells (**BIN**), the amount of solar cells used to make the solar module (**COUNT_CELLS**) and a flag to understand what the minimum class is of the solar cells (**MIN_BIN**). However this was added only for test purposes.

Considering the same dataset and predictions, every single solar module can be summarised in one row, saving the Pmpps predicted (Minimum and Maximum) in relation to the two Solar Cells classes (BINs) and the count of solar cells for each class. As Table 8.7 shows, the Pmpp(s) predicted seem to correspond to the minimum and maximum value of the interval where the Pmpp target seems to remain in the middle of this interval. In addition, we discovered that the minimum solar cell class (BIN) affects the final Pmpp more than the maximum solar cell class (about **60%**). Therefore, definition of a new weighted Pmpp predicted using the two Pmpp predicted before with some weights is needed.

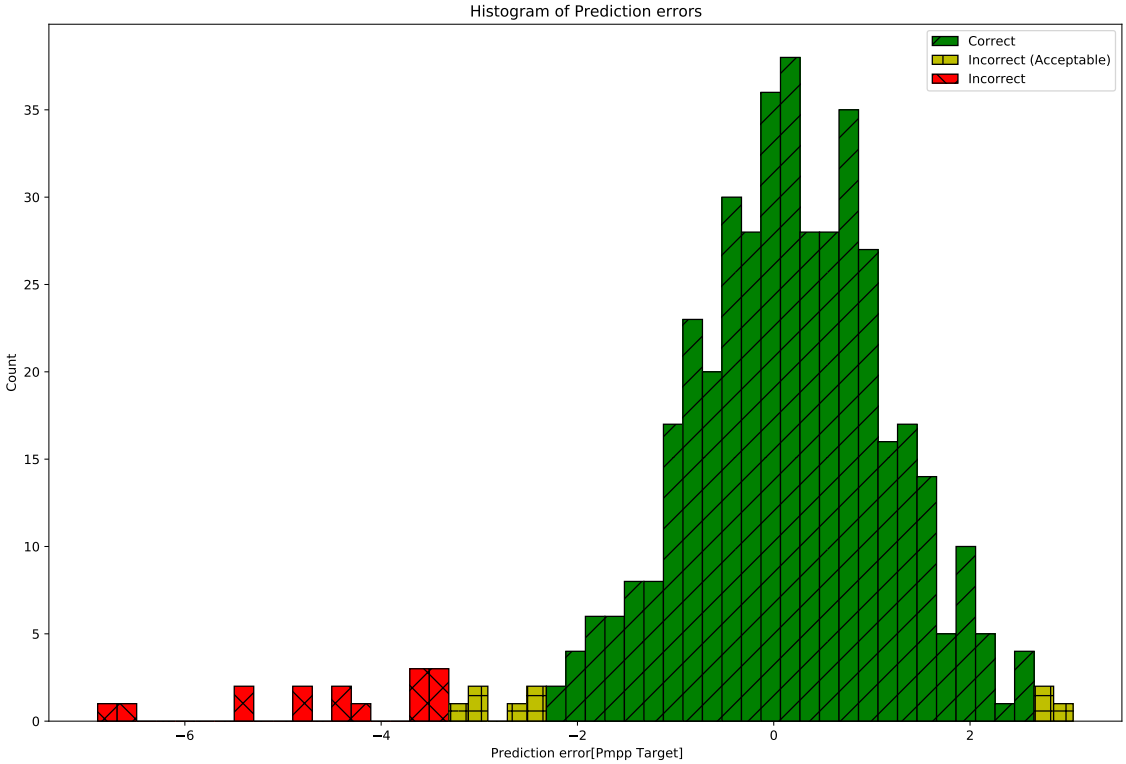


Figure 8.6: Distributions of prediction results (with two BINs)

Let $Weight_{min}$ and $Weight_{max}$ be two constant weights applied for their respective minimum and maximum classes:

$$\begin{gathered} \text{Let } Weight_{min} = 0.6 \text{ and } Weight_{max} = 0.4\ ; \\ \mu = (CountCells_{max} \cdot Weight_{max}); \\ \nu = (CountCells_{min} \cdot Weight_{min}); \\ \text{then define a new Pmpp Weighted as follows:} \end{gathered} \tag{8.13}$$

$$PmppPredicted_{weighted} = \frac{((PmppPredicted_{min} \cdot \nu) + (PmppPredicted_{max} \cdot \mu))}{\nu + \mu} \tag{8.14}$$

As test dataset, the dataset used consisted of **440** solar modules made of two classes of solar cells. The predictions results, as Figure 8.6 shows, matched **415** modules of 440 given, with **25** incorrect predictions. Therefore, the model achieves

	Pmpp Target	**Pmpp Predicted Weighted**	**Error Weighted**
count	440.000000	440.000000	440.000000
mean	375.558011	375.555437	0.002574
std	4.051693	3.706731	1.334370
min	353.620209	353.127561	-6.888446
25%	373.826920	373.956401	-0.593594
50%	375.897400	375.865245	0.128161
75%	378.197914	377.818066	0.813447
max	385.207001	383.648547	3.050453

Table 8.8: Prediction results statistics (with two BINs)

an accuracy of prediction about **94.32%** (only the Corrected prediction were considered). Another aspect that gives good results is the statistical information about the errors. The predictions results demonstrates that the neural network model can predict the final maximum power point of a solar module made with one or two solar cell classes, with a high accuracy of prediction.

Considering the theoretical aspect, as described above, we would have expected the final power attested to the power linked to the electrical parameters of the cells less efficient, while instead, it seems that there is a power gain due to cells with better efficiency[4].

8.7 Application

Since the model was developed using the Keras framework, its usage could be problematic. Therefore, a server/desktop application was made. The application was developed as a multi-platform application, which means the application can run in a Linux or Windows environment (maybe also in macOS). The Electron framework[5] was chosen to make the final application. The framework allows the development of the application using Javascript language, through a Javascript interpreter, such as WebKit.

The graphical user interface was limited to the design and the development of a Web interface, within which the users can use the model. The original model was

[4]The modules used in these experiments are currently under reliability testing to verify that the use of larger classes do not compromise the quality of the final product.

[5]https://www.electronjs.org/

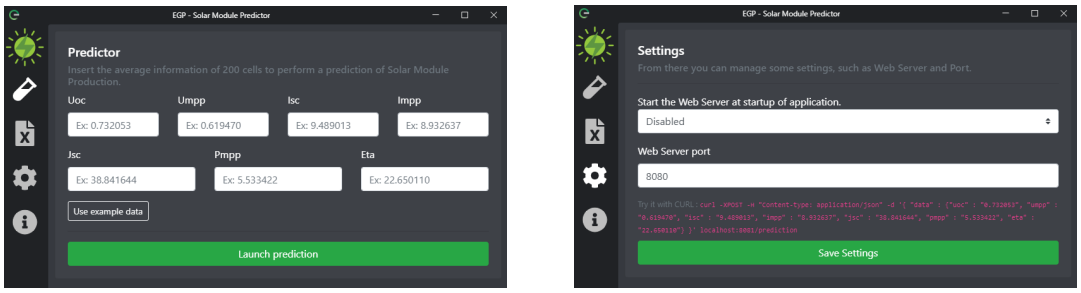


Figure 8.7: Some application screens.

converted into the TensorFlowJS type that can be used with the final application. The application offers some simple views where the users can operate with the model. In the first view, the users can use the model passing the relative inputs and perform the prediction. The second view allows the users to load a CSV file with more than one row which will be used to make more than one prediction. Before the launch of the predictions, the input file will be analysed, verifying the integrity and format of the file and the column labels. If the labels do not respect the requirements, the software will ask the user to select the correct columns. The third view allows the users (or administrators in this case) to launch a simple HTTP server, specifying the service port, which will be served to allow the prediction service via the network. The last view contains some information about the developer and the software version. A new version of the software was made by using GoLang language[6]. Nevertheless, the resulting application can run only in a Linux environment, therefore was scrapped despite its performances[7].

8.8 Conclusions

In this chapter we introduced the physical basis of solar cells and solar modules, introducing the I-V characteristic of the solar cells and, therefore, the problem of the final power produced by the solar modules in the real world. Furthermore, the real problem about the power produced is stronger. Part of this problem is the automated manufacturing environment and settings, where the "trial and error"

[6] https://golang.org/

[7] Also the installation process would be more difficult than the Electron version

has a cost. In addition, the assembly phases of solar cells and solar modules could be improved, for instance by improving the classification system of the cells and reducing the number of the classes.

Therefore, we created a neural network model which can predict the final maximum power of a solar module, made with a set of solar cells as inputs. The proposed model works fine with one or two types of solar cell classes. This permits new types of experiments such as exploring new recipes or new solar cell classes. In addition, the accuracy of the proposed model seems acceptable with about 94% of accuracy and finally, it can be used to simulate the production of the final maximum power of solar modules without any production costs.

For future research, the model must be more generic, because, the model can only predict the power of the solar module with one or two different classes of solar cells. Therefore, a new model must be trained with a new type of dataset and should be designed to be generic. Finally, the results of the predictions should be demonstrated, making real solar modules with the same set of solar cells used for prediction and compare the results.

8.9 Acknowledgments

We thank Dario Iuvara[8] and Fabrizio Bizzarri[9] for the help and collaboration given to us during our research and work.

[8]Head of the Automation Innovation Support Unit of Enel Green Power (dario.iuvara@enel.com)
[9]Head of the Solar Innovation Unit of Enel Green Power (fabrizio.bizzarri@enel.com)

Chapter 9

Conclusions

The works presented in this thesis aim to advance the state of the art in Industrial IoT and represent a step forward in the implementation and expansion of IoT technology in any field. The presented approaches can be used as a guide for the creation of new protocols, simulators, elaborations, and prediction systems to improve industrial technologies.

In the first part, this thesis addresses on communication protocols and privacy preservation of IoT. Furthermore, a new distributed platform of the UAV flock control, a new communication protocol which allows the automatic configuration of the devices' network and a new routing protocol for the DTN networks have been discussed.

In the second part, this thesis introduces a new UAV simulator which can also emulate wireless networks and an improvement of Docker technology. The latter allows the sharing of libraries between containers.

Finally, in the third part, the thesis regards the artificial intelligence application on IoT. This introduces a new smart social agent which enables the interactions between humans and domotic devices through social networks and a neural network model capable of predicting the efficiency of a solar module before its creation.

The results of each chapter demonstrate the application feasibility of each proposed solution. Moreover, the same results provide new ideas for future works, useful to further improve the Internet of Things.

Bibliography

[1] K. Ashton et al. "That 'internet of things' thing". In: *RFID journal* 22.7 (2009), pp. 97–114.

[2] M. Zwolenski, L. Weatherill, et al. "The digital universe: Rich data and the increasing value of the internet of things". In: *Journal of Telecommunications and the Digital Economy* 2.3 (2014), p. 47.

[3] C. Counter. "The Internet of Everything in Motion". In: *Cisco's Technology News Site.—[Elec tronic resource].—Mode of Access: https://newsroom. cisco. com/featurecontent* ().

[4] J. Rivera and R. van der Meulen. "Gartner says the internet of things installed base will grow to 26 billion units by 2020". In: *Stamford, conn., December* 12 (2013).

[5] G. E. Moore. "Cramming more components onto integrated circuits". In: *Proceedings of the IEEE* 86.1 (1998), pp. 82–85.

[6] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. "Assessing trends in the electrical efficiency of computation over time". In: *IEEE Annals of the History of Computing* 17 (2009).

[7] L. Atzori, A. Iera, and G. Morabito. "Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm". In: *Ad Hoc Networks* 56 (2017), pp. 122–140.

[8] J. Zheng and M. J. Lee. "A comprehensive performance study of IEEE 802.15. 4". In: *Sensor network operations* 4 (2006), pp. 218–237.

[9] M. H. Coen et al. "Design principles for intelligent environments". In: *AAAI/IAAI*. 1998, pp. 547–554.

[10] M. Hoeynck and B. W. Andrews. *Sensor-Based Occupancy and Behavior Prediction Method for Intelligently Controlling Energy Consumption Within a Building*. US Patent App. 12/183,361. 2010.

[11] C Gomez-Otero, R Martinez, and J Caffarel. "ClimApp: A novel approach of an intelligent HVAC control system". In: *7th Iberian Conference on Information Systems and Technologies (CISTI 2012)*. IEEE. 2012, pp. 1–6.

[12] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. "Industry 4.0". In: *Business & information systems engineering* 6.4 (2014), pp. 239–242.

[13] A. Gilchrist. *Industry 4.0: the industrial internet of things*. Springer, 2016.

[14] K. Witkowski. "Internet of things, big data, industry 4.0–innovative solutions in logistics and supply chains management". In: *Procedia Engineering* 182 (2017), pp. 763–769.

[15] M. Wollschlaeger, T. Sauter, and J. Jasperneite. "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0". In: *IEEE industrial electronics magazine* 11.1 (2017), pp. 17–27.

[16] D. Bandyopadhyay and J. Sen. "Internet of things: Applications and challenges in technology and standardization". In: *Wireless personal communications* 58.1 (2011), pp. 49–69.

[17] F. Shrouf, J. Ordieres, and G. Miragliotta. "Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm". In: *2014 IEEE international conference on industrial engineering and engineering management*. IEEE. 2014, pp. 697–701.

[18] F. D'Urso, C. Santoro, and F. F. Santoro. "Integrating Heterogeneous Tools for Physical Simulation of multi-Unmanned Aerial Vehicles." In: *WOA*. 2018, pp. 10–15.

[19] F. D'Ursol, C. Grasso, C. Santoro, F. F. Santoro, and G. Schembra. "The Tactile Internet for the flight control of UAV flocks". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE. 2018, pp. 470–475.

[20] C. Santoro, F. Messina, F. D'Urso, and F. F. Santoro. "Wale: a Dockerfile-based approach to deduplicate shared libraries in Docker containers". In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE. 2018, pp. 785–791.

[21] F. D'Urso, C. Santoro, and F. F. Santoro. "An integrated framework for the realistic simulation of multi-UAV applications". In: *Computers & Electrical Engineering* 74 (2019), pp. 196–209.

[22] C. F. Longo, C. Santoro, and F. F. Santoro. "Meaning Extraction in a Domotic Assistant Agent Interacting by means of Natural Language". In: *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE. 2019, pp. 21–26.

[23] M. Buffa, F. Messina, C. Santoro, and F. F. Santoro. "Design of Self-organizing Protocol for LoWPAN Networks". In: *International Conference on Internet and Distributed Computing Systems*. Springer. 2019, pp. 424–433.

[24] F. D'Urso, C. Santoro, and F. F. Santoro. "Wale: A solution to share libraries in Docker containers". In: *Future Generation Computer Systems* 100 (2019), pp. 513–522.

[25] M. De Benedetti, F. D'Urso, G. Fortino, F. Messina, G. Pappalardo, and C. Santoro. "A Fault-tolerant Self-organizing Flocking Approach for UAV Aerial Survey". In: *J. Netw. Comput. Appl.* 96.C (Oct. 2017), pp. 14–30. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2017.08.004. URL: https://doi.org/10.1016/j.jnca.2017.08.004.

[26] Z. Zheng, A. K. Sangaiah, and T. Wang. "Adaptive Communication Protocols in Flying Ad Hoc Network". In: *IEEE Communications Magazine* 56.1 (2018), pp. 136–142. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1700323.

[27] G. S. C. Rametta. ""Designing a softwarized network deployed on a fleet of drones for rural zone monitoring"". In: *Future Internet* 9.1 (2017).

[28] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van. "The tactile internet: vision, recent progress, and open challenges". In: *IEEE Communications Magazine* 54.5 (2016), pp. 138–145.

[29] A. A. Ateya, A. Muthanna, and A. Koucheryavy. "5G framework based on multi-level edge computing with D2D enabled communication". In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. 2018, pp. 507–512. DOI: 10.23919/ICACT.2018.8323812.

[30] S. Iellamo, J. J. Lehtomaki, and Z. Khan. "Placement of 5G Drone Base Stations by Data Field Clustering". In: *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*. 2017, pp. 1–5.

[31] L. Chiaraviglio, W. Liu, J. A. Gutierrez, and N. Blefari-Melazzi. "Optimal pricing strategy for 5G in rural areas with unmanned aerial vehicles and large cells". In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017, pp. 1–7. DOI: 10.1109/ATNAC.2017.8215406.

[32] M. Dohler, T. Mahmoodi, M. A. Lema, M. Condoluci, F. Sardis, K. Antonakoglou, and H. Aghvami. "Internet of skills, where robotics meets AI, 5G and the Tactile Internet". In: *2017 European Conference on Networks and Communications (EuCNC)*. 2017, pp. 1–5. DOI: 10.1109/EuCNC.2017.7980645.

[33] A. Aijaz, M. Dohler, A. H. Aghvami, V. Friderikos, and M. Frodigh. "Realizing the Tactile Internet: Haptic Communications over Next Generation 5G Cellular Networks". In: *IEEE Wireless Communications* 24.2 (2017), pp. 82–89. ISSN: 1536-1284. DOI: 10.1109/MWC.2016.1500157RP.

[34] F. Sophia Antipolis. *Mobile edge computing: A key technology towards 5G*. 2015.

[35] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration". In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1657–1681. DOI: 10.1109/COMST.2017.2705720.

[36] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999.

[37] J. d. J. Gil Herrera and J. F. B. Vega. "Network Functions Virtualization: A Survey". In: *IEEE Latin America Transactions* 14.2 (2016), pp. 983–997. ISSN: 1548-0992. DOI: 10.1109/TLA.2016.7437249.

[38] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. "Network Slicing in 5G: Survey and Challenges". In: *IEEE Communications Magazine* 55.5 (2017), pp. 94–100. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600951.

[39] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira. "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges". In: *IEEE Communications Magazine* 55.5 (2017), pp. 80–87. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600935.

[40] J.-C. Latombe. "Exact Cell Decomposition". In: *Robot Motion Planning*. Springer, 1991, pp. 200–247.

[41] J.-C. Latombe. "Approximate cell decomposition". In: *Robot Motion Planning*. Springer, 1991, pp. 248–294.

[42] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset. "Efficient boustrophedon multi-robot coverage: an algorithmic approach". In: *Annals of Mathematics and Artificial Intelligence* 52.2-4 (2008), pp. 109–142.

[43] M. De Benedetti, F. D'Urso, F. Messina, G. Pappalardo, and C. Santoro. "3D Simulation of Unmanned Aerial Vehicles". In: *XVIII Workshop "Dagli Oggetti agli Agenti"*. CEUR-WS. 2017.

[44] P. Pace, G. Aloi, G. Caliciuri, and G. Fortino. "A mission-oriented coordination framework for teams of mobile aerial and terrestrial smart objects". In: *Mobile Networks and Applications* 21.4 (2016), pp. 708–725.

[45] M. D. Benedetti, F. D'Urso, F. Messina, G. Pappalardo, and C. Santoro. "UAV-based Aerial Monitoring: A Performance Evaluation of a Self-Organising Flocking Algorithm". In: *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. 2015, pp. 248–255. DOI: 10.1109/3PGCIC.2015.78.

[46] L. Atzori, A. Iera, and G. Morabito. "The internet of things: A survey". In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[47] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.

[48] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. "Internet of things for smart cities". In: *IEEE Internet of Things journal* 1.1 (2014), pp. 22–32.

[49] A. Gaur, B. Scotney, G. Parr, and S. McClean. "Smart city architecture and its applications based on IoT". In: *Procedia computer science* 52 (2015), pp. 1089–1094.

[50] A. Dunkels, B. Gronvall, and T. Voigt. "Contiki-a lightweight and flexible operating system for tiny networked sensors". In: *29th annual IEEE international conference on local computer networks*. IEEE. 2004, pp. 455–462.

[51] G. Mulligan. "The 6LoWPAN architecture". In: *Proceedings of the 4th workshop on Embedded networked sensors*. ACM. 2007, pp. 78–82.

[52] N. Accettura, L. A. Grieco, G. Boggia, and P. Camarda. "Performance analysis of the RPL routing protocol". In: *2011 IEEE International Conference on Mechatronics*. IEEE. 2011, pp. 767–772.

[53] C. Bormann, A. P. Castellani, and Z. Shelby. "Coap: An application protocol for billions of tiny internet nodes". In: *IEEE Internet Computing* 2 (2012), pp. 62–67.

[54] A. Dunkels. "The contikimac radio duty cycling protocol". In: (2011).

[55] S. Kalyoncu. *Wireless solutions and authentication mechanisms for Contiki based Internet of things networks*. 2013.

[56] H.-S. Kim, S. Kumar, and D. E. Culler. "Thread/OpenThread: A Compromise in Low-Power Wireless Multihop Network Architecture for the Internet of Things". In: *IEEE Communications Magazine* (2019), p. 3.

[57] G. Inc. *OpenThread*. https://openthread.io/. 2019.

[58] T. T. Group. *Thread*. https://www.threadgroup.org/. 2019.

[59] R. Rajsuman. *System-on-a-chip: Design and Test*. Artech House, Inc., 2000.

[60] R. Giladi. *Network processors: architecture, programming, and implementation*. Morgan Kaufmann, 2008.

[61] N Sornin, M Luis, T Eirich, T Kramp, and O Hersent. "LoRaWAN specification". In: *LoRa alliance* (2015).

[62] J. de Carvalho Silva, J. J. Rodrigues, A. M. Alberti, P. Solic, and A. L. Aquino. "LoRaWAN—A low power WAN protocol for Internet of Things: A review and opportunities". In: *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*. IEEE. 2017, pp. 1–6.

[63] N. Kushalnagar and G. Montenegro. "Transmission of IPv6 packets over IEEE 802.15. 4 networks". In: (2007).

[64] A. Zacepins, A. Kviesis, P. Ahrendt, U. Richter, S. Tekin, and M. Durgun. "Beekeeping in the future—Smart apiary management". In: *2016 17th International Carpathian Control Conference (ICCC)*. IEEE. 2016, pp. 808–812.

[65] K. Fall. "A delay-tolerant network architecture for challenged internets". In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM. 2003, pp. 27–34.

[66] K. L. Scott. "Bundle protocol specification". In: (2007).

[67] T. Small and Z. J. Haas. "Resource and performance tradeoffs in delay-tolerant wireless networks". In: *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. ACM. 2005, pp. 260–267.

[68] A. Vahdat, D. Becker, et al. "Epidemic routing for partially connected ad hoc networks". In: (2000).

[69] L Wood, J McKim, W Eddy, W Ivancic, and C Jackson. "Saratoga: A scalable file transfer protocol". In: *work in progress as an Internet-draft* (2010).

[70] A. Lindgren, A. Doria, and O. Schelén. "Probabilistic routing in intermittently connected networks". In: *ACM International Symposium on Mobilde Ad Hoc Networking and Computing, MobiHoc 2003: 01/06/2003-03/06/2003*. 2003.

[71] Y. Lindell and B. Pinkas. "Privacy preserving data mining". In: *Annual International Cryptology Conference*. Springer. 2000, pp. 36–54.

[72] C. Clifton, M. Kantarcioglu, and J. Vaidya. "Defining privacy for data mining". In: *National science foundation workshop on next generation data mining*. Vol. 1. 26. Citeseer. 2002, p. 1.

[73] Y. Zhu and Y. Hu. "Making peer-to-peer anonymous routing resilient to failures". In: *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2007, pp. 1–10.

[74] E. Papapetrou, V. F. Bourgos, and A. G. Voyiatzis. "Privacy-preserving routing in delay tolerant networks based on bloom filters". In: *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE. 2015, pp. 1–9.

[75] P. Hui, J. Crowcroft, and E. Yoneki. "Bubble rap: Social-based forwarding in delay-tolerant networks". In: *IEEE Transactions on Mobile Computing* 10.11 (2010), pp. 1576–1589.

[76] E. M. Daly and M. Haahr. "Social network analysis for routing in disconnected delay-tolerant manets". In: *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*. 2007, pp. 32–40.

[77] A. Guellier, C. Bidan, and N. Prigent. "Homomorphic cryptography-based privacy-preserving network communications". In: *International Conference on Applications and Techniques in Information Security*. Springer. 2014, pp. 159–170.

[78] A. V. Vasilakos, Y. Zhang, and T. Spyropoulos. *Delay tolerant networks: Protocols and applications*. CRC press, 2016.

[79] A. Lindgren, E. Davies, S. Grasic, and A. Doria. "Probabilistic routing protocol for intermittently connected networks". In: (2012).

[80] R. W. Schafer and L. R. Rabiner. "Digital representations of speech signals". In: *Proceedings of the IEEE* 63.4 (1975), pp. 662–677.

[81] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*. Vol. 37. Springer Science & Business Media, 2010.

[82] S. McKinley and M. Levine. "Cubic spline interpolation". In: *College of the Redwoods* 45.1 (1998), pp. 1049–1060.

[83] G. Cai, J. Dias, and L. Seneviratne. "A survey of small-scale unmanned aerial vehicles: Recent advances and future development trends". In: *Unmanned Systems* 2.02 (2014), pp. 175–199.

[84] Y. Wei, M. B. Blake, and G. R. Madey. "An operation-time simulation framework for UAV swarm configuration and mission planning". In: *Procedia Computer Science* 18 (2013), pp. 1949–1958.

[85] T. R. F. Cavalcante, I. V. d. Bessa, and L. C. Cordeiro. "Planning and Evaluation of UAV Mission Planner for Intralogistics Problems". In: *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2017, pp. 9–16.

[86] G. Chmaj and H. Selvaraj. "Distributed processing applications for UAV/-drones: a survey". In: *Progress in Systems Engineering*. Springer, 2015, pp. 449–454.

[87] C. Virágh et al. "Flocking algorithm for autonomous flying robots". In: *Bioinspiration & biomimetics* 9.2 (2014), p. 025012.

[88] N. Bouraqadi and A. Doniec. "Flocking-Based Multi-Robot Exploration". In: *Proceedings of the* 4^{th} *National Conference on Control Architectures of Robots*. Toulouse, France, 2009.

[89] G. Vásárhelyi et al. "Outdoor flocking and formation flight with autonomous aerial robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IROS '14. Chicago, IL, USA, 2014, pp. 3866–3873.

[90] E. Sklar. "Software Review: NetLogo, a Multi–agent Simulation Environment". In: *Artificial Life* 13 (2007), pp. 303–311.

[91] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. "MASON: a Multi–Agent Simulation Environment". In: *Simulation* 81.7 (2005), pp. 517–527.

[92] E. Tatara, M. North, T. Howe, N. T. Collier, and J. Vos. "An introduction to Repast modelling using a simple predator–prey example". In: *Proceedings of Agents 2006 Conference on Social Agents: Results and Prospects*. 2006.

[93] N. Koenig and A. Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *International Conference on Intelligent Robots and Systems*. Sendai, Japan, 2004, pp. 2149–2154.

[94] G. F. Riley and T. R. Henderson. "The ns-3 Network Simulator". In: *Modeling and Tools for Network Simulation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. ISBN: 978-3-642-12331-3.

[95] A. Varga and R. Hornig. "An overview of the OMNeT++ simulation environment". In: *In Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*. 2008.

[96] A. G. Madey and G. R. Madey. "Design and Evaluation of UAV Swarm Command and Control Strategies". In: *Proceedings of the Agent-Directed Simulation Symposium*. ADSS 13. San Diego, California: Society for Computer Simulation International, 2013, 7:1–7:8. ISBN: 978-1-62748-029-1.

[97] S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J. Zufferey, and D. Floreano. "Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*. 2011, pp. 5015–5020.

[98] L. Ciarletta, A. Guenard, Y. Presse, V. Galtier, Y. Q. Song, J. C. Ponsart, S. Aberkane, and D. Theilliol. "Simulation and platform tools to develop safe flock of UAVs: a CPS application-driven research". In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014, pp. 95–102.

[99] E. A. Marconato, M. Rodrigues, R. de Melo Pires, D. F. Pigatto, L. C. Q. Filho, A. R. Pinto, and K. R. L. J. C. Branco. "AVENS - A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System". In: *HICSS*. 2017.

[100] M. Kudelski, L. M. Gambardella, and G. A. Di Caro. "RoboNetSim: An Integrated Framework for Multi-robot and Network Simulation". In: *Robot. Auton. Syst.* 61.5 (May 2013), pp. 483–496. ISSN: 0921-8890.

[101] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe. "Co-simulation: State of the art". In: *CoRR* abs/1702.00686 (2017). arXiv: 1702.00686.

[102] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. "RotorS—A modular gazebo MAV simulator framework". en. In: *Robot operating system (ROS) : the complete reference (volume 1)*. Ed. by A. Koubaa. Vol. 625. Studies in computational intelligence. . Springer, 2016, pp. 595 –625. ISBN: 978-3-319-26054-9.

[103] L. Meier, D. Honegger, and M. Pollefeys. "PX4: A Node-Based Multithreaded Open Source Robotics Framework for Deeply Embedded Platforms". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. 2015.

[104] M. Kudelski, M. Cinus, L. Gambardella, and G. A. Di Caro. "A framework for realistic simulation of networked multi-robot systems". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5018–5025.

[105] S. Nethi, M. Pohjola, L. Eriksson, and R. Jantti. "Platform for emulating networked control systems in laboratory environments". In: *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*. IEEE. 2007, pp. 1–8.

[106] S. Hayat, E. Yanmaz, and R. Muzaffar. "Survey on Unmanned Aerial Vehicle Networks for Civil Applications: A Communications Viewpoint". In: *IEEE Communications Surveys Tutorials* 18.4 (2016), pp. 2624–2661. ISSN: 1553-877X.

[107] M. Schmittle, A. Lukina, L. Vacek, J. Das, C. Buskirk, S. Rees, J. Sztipanovits, R. Grosu, and V. Kumar. "OpenUAV: A UAV Testbed for the CPS and Robotics Community". In: *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. ICCPS '18. Porto, Portugal: IEEE Press, 2018, pp. 130–139. ISBN: 978-1-5386-5301-2.

[108] C. Reynolds. "Flocks, Herds and Schools: A Distributed Behavioral Model". In: *Proceedings of the* 14^{th} *Annual Conference on Computer Graphics and*

Interactive Techniques. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34.

[109] P. Mell and T. Grance. "The NIST definition of cloud computing". In: *Communications of the ACM* 53.6 (2010), p. 50.

[110] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. "Xen and the art of virtualization". In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM. 2003, pp. 164–177.

[111] P. Dash. *Getting started with Oracle VM VirtualBox*. Packt Publishing Ltd, 2013.

[112] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith. "Intel virtualization technology". In: *Computer* 38.5 (2005), pp. 48–56.

[113] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. "kvm: the Linux virtual machine monitor". In: *Proceedings of the Linux symposium*. Vol. 1. 2007, pp. 225–230.

[114] C. D. Graziano. "A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project". In: (2011).

[115] A. Desai, R. Oza, P. Sharma, and B. Patel. "Hypervisor: A survey on concepts and taxonomy". In: *International Journal of Innovative Technology and Exploring Engineering* 2.3 (2013), pp. 222–225.

[116] P. M. Perera and C. Keppitiyagama. "A performance comparison of hypervisors". In: *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference on*. IEEE. 2011, pp. 120–120.

[117] D. Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (2014), p. 2.

[118] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[119] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. "Microservices: yesterday, today, and tomorrow". In: *Present and Ulterior Software Engineering*. Springer, 2017, pp. 195–216.

[120] S. Newman. *Building microservices: designing fine-grained systems.* "O'Reilly Media, Inc.", 2015.

[121] A. H. Sodhro, S. Pirbhulal, M. Qaraqe, S. Lohano, G. H. Sodhro, N. U. R. Junejo, and Z. Luo. "Power control algorithms for media transmission in remote healthcare systems". In: *IEEE Access* 6 (2018), pp. 42384–42393.

[122] A. H. Sodhro, Z. Luo, A. K. Sangaiah, and S. W. Baik. "Mobile edge computing based QoS optimization in medical healthcare applications". In: *International Journal of Information Management* (2018). ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2018.08.004.

[123] A. H. Sodhro, S. Pirbhulal, and A. K. Sangaiah. "Convergence of IoT and product lifecycle management in medical health care". In: *Future Generation Computer Systems* 86 (2018), pp. 380 –391. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.03.052.

[124] A. Sodhro, A. Sangaiah, G. Sodhro, S. Lohano, and S. Pirbhulal. "An energy-efficient algorithm for wearable electrocardiogram signal processing in ubiquitous healthcare applications". In: *Sensors* 18.3 (2018), p. 923.

[125] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. "Difference engine: Harnessing memory redundancy in virtual machines". In: *Communications of the ACM* 53.10 (2010), pp. 85–93.

[126] A. Arcangeli, I. Eidus, and C. Wright. "Increasing memory density by using KSM". In: *Proceedings of the linux symposium*. Citeseer. 2009, pp. 19–28.

[127] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman. "Satori: Enlightened page sharing". In: *Proceedings of the 2009 conference on USENIX Annual technical conference*. 2009, pp. 1–1.

[128] K. Suzaki, T. Yagi, K. Iijima, A.-Q. Nguyen, C. Artho, and Y. Watanebe. "Moving from Logical Sharing of Guest OS to Physical Sharing of Deduplication on Virtual Machine." In: *HotSec*. 2010.

[129] K. Suzaki, K. Iijima, T. Yagi, and C. Artho. "Memory deduplication as a threat to the guest OS". In: *Proceedings of the Fourth European Workshop on System Security*. ACM. 2011, p. 1.

[130] K. Jin and E. L. Miller. "The effectiveness of deduplication on virtual machine disk images". In: *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM. 2009, p. 7.

[131] O. Rodeh, J. Bacik, and C. Mason. "BTRFS: The Linux B-tree filesystem". In: *ACM Transactions on Storage (TOS)* 9.3 (2013), p. 9.

[132] O. Rodeh and A. Teperman. "zFS-a scalable distributed file system using object disks". In: *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*. IEEE. 2003, pp. 207–218.

[133] J. Bonwick and B. Moore. "ZFS: The last word in file systems". In: (2007).

[134] H.-f. XUE, S.-h. QING, and H.-g. ZHANG. "Analysis on XEN virtualization machine [J]". In: *Journal of System Simulation* 23 (2007), p. 052.

[135] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. C. Fox. "GPU passthrough performance: A comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL applications". In: *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE. 2014, pp. 636–643.

[136] J. Torbić, I. Stanković, B. Đorđević, and V. Timčenko. "Hyper-V and ESXi hypervisors comparison in Windows Server 12 virtual environment". In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. 2018, pp. 1–5. DOI: 10.1109/INFOTEH.2018.8345548.

[137] V. K. Manik and D. Arora. "Performance comparison of commercial VMM: ESXI, XEN, HYPER-V, KVM". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2016, pp. 1771–1775.

[138] D. J. Magenheimer, P. Ranganathan, and M. Chapman. *Virtualization with binary translation*. US Patent 8,327,354. 2012.

[139] B.-C. Le. *Emulation system that uses dynamic binary translation and permits the safe speculation of trapping operations*. US Patent 6,631,514. 2003.

[140] E. R. Altman, K. Ebcioglu, M. Gschwind, and S. Sathaye. "Advances and future challenges in binary translation and optimization". In: *Proceedings of the IEEE* 89.11 (2001), pp. 1710–1722.

[141] G. K. Thiruvathukal, K. Hinsen, K. Läufer, and J. Kaylor. "Virtualization for computational scientists". In: *Computing in Science & Engineering* 12.4 (2010), pp. 52–61.

[142] P. Li. "Selecting and using virtualization solutions: our experiences with VMware and VirtualBox". In: *Journal of Computing Sciences in Colleges* 25.3 (2010), pp. 11–17.

[143] S. N. T.-c. Chiueh and S. Brook. "A survey on virtualization technologies". In: *Rpe Report* 142 (2005).

[144] B. des Ligneris. "Virtualization of Linux based computers: the Linux-VServer project". In: *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*. IEEE. 2005, pp. 340–346.

[145] F. Bellard. "QEMU, a fast and portable dynamic translator." In: *USENIX Annual Technical Conference, FREENIX Track*. Vol. 41. 2005, p. 46.

[146] D. Quigley, J. Sipek, C. P. Wright, and E. Zadok. "Unionfs: User-and community-oriented development of a unification filesystem". In: *Proceedings of the 2006 Linux Symposium*. Vol. 2. 2006, pp. 349–362.

[147] B. L. R. Stojkoska and K. V. Trivodaliev. "A review of Internet of Things for smart home: Challenges and solutions". In: *Journal of Cleaner Production* 140 (2017), pp. 1454–1464.

[148] K.-L. Tsai, F.-Y. Leu, and I. You. "Residence energy control system based on wireless smart socket and IoT". In: *IEEE Access* 4 (2016), pp. 2885–2894.

[149] *IFTTT: If This Then That.*

[150] M. Mehrabani, S. Bangalore, and B. Stern. "Personalized speech recognition for Internet of Things". In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, pp. 369–374. DOI: 10.1109/WF-IoT.2015.7389082.

[151] S. T. Manfred Eppe and J. Feldman. "Exploiting Deep Semantics and Compositionality of Natural Language for Human-Robot Interaction". In: *International Conference on Intelligence Robots and System in Daejeon, Korea*. IEEE/RSJ. 2016.

[152] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banarjee, S. Teller, and N. Roy. "Understanding natural language commands for robotic navigation and mobile manipulation". In: *Conference on Artificial Intelligence*. AAAI. 2011.

[153] G.-J. M. Kruij, P. Lison, T. Benjamin, H. Jacobsson, H. Zender, and I. Kruij-Korbayov. "Situated Dialogue Processing for Human-Robot Interaction". In: *Cognitive Systems* (2010).

[154] G.-J. M. Kruijff, H. Zender1, P. Jensfelt, and H. I. Christensen. "Situated Dialogue and Spatial Organization: What, Where... and Why?" In: *International Journal of Advanced Robotic Systems* (2007).

[155] M. Steedman. "The syntactic process". In: *MIT Press* (2000).

[156] W. F. T. Mathias Landhäußer Sebastian Weigelt. "NLCI: a natural language command interpreter". In: *Springer Science+Business Media New York* (2016).

[157] L. Fichera, F. Messina, G. Pappalardo, and C. Santoro. "A Python framework for programming autonomous robots using a declarative approach". In: *Sci. Comput. Program.* 139 (2017), pp. 36–55. DOI: 10.1016/j.scico.2017.01.003. URL: https://doi.org/10.1016/j.scico.2017.01.003.

[158] H. Matthew. *spaCy: Industrial-Strength Natural Language Processing*. https://spacy.io. 2017.

[159] C. D. Manning, MihaiSurdeanu, J. Bauer, J. R. Finkel, StevenBethard, and D. McClosk. "The stanford corenlp natural language processing toolki". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 55-60*. 2014.

[160] G. A. Miller. "WordNet: A Lexical Database for English". In: *Communications of the ACM Vol. 38, No. 11: 39-41*. 1995.

[161] *CMUSphinx: Open-source speech recognition toolkit*.

[162] A. Bartler, L. Mauch, B. Yang, M. Reuter, and L. Stoicescu. "Automated Detection of Solar Cell Defects with Deep Learning". In: *2018 26th European Signal Processing Conference (EUSIPCO)*. 2018, pp. 2035–2039. DOI: 10.23919/EUSIPCO.2018.8553025.

[163] V. Golovko, S. Bezobrazov, A. Kroshchanka, A. Sachenko, M. Komar, and A. Karachka. "Convolutional neural network based solar photovoltaic panel detection in satellite photos". In: *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Vol. 1. 2017, pp. 14–19. DOI: 10.1109/IDAACS.2017.8094501.

[164] K. Jazayeri, S. Uysal, and M. Jazayeri. "Determination of power losses in solar panels using artificial neural network". In: *2013 Africon*. 2013, pp. 1–6. DOI: 10.1109/AFRCON.2013.6757775.

[165] D. D. Nguyen, B. Lehman, and S. Kamarthi. "Performance evaluation of solar photovoltaic arrays including shadow effects using neural network". In: *2009 IEEE Energy Conversion Congress and Exposition*. 2009, pp. 3357–3362. DOI: 10.1109/ECCE.2009.5316451.

[166] T. Verma, A. P. S. Tiwana, C. C. Reddy, V. Arora, and P. Devanand. "Data Analysis to Generate Models Based on Neural Network and Regression for Solar Power Generation Forecasting". In: *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*. 2016, pp. 97–100. DOI: 10.1109/ISMS.2016.65.

[167] M. Taherbaneh and K. Faez. "Maximum Power Point Estimation for Photovoltaic Systems Using Neural Networks". In: *2007 IEEE International Conference on Control and Automation*. 2007, pp. 1614–1619. DOI: 10.1109/ICCA.2007.4376633.

[168] M. Green, A. Blakers, S Narayanan, and M Taouk. "Improvements in silicon solar cell efficiency". In: *Solar cells* 17.1 (1986), pp. 75–83.

[169] S. Katsikeas, K. Fysarakis, A. Miaoudakis, A. Van Bemten, I. Askoxylakis, I. Papaefstathiou, and A. Plemenos. "Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol". In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2017, pp. 1193–1200.

[170] A. Xenakis, A. Karageorgos, E. Lallas, A. E. Chis, and H. González-Vélez. "Towards distributed IoT/cloud based fault detection and maintenance in industrial automation". In: *Procedia Computer Science* 151 (2019), pp. 683–690.

[171] M. A. Green, E. D. Dunlop, D. H. Levi, J. Hohl-Ebinger, M. Yoshita, and A. W. Ho-Baillie. "Solar cell efficiency tables (version 54)". In: *Progress in Photovoltaics: Research and Applications* 27.7 (2019), pp. 565–575. DOI: https://doi.org/10.1002/pip.3171. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pip.3171. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pip.3171.

[172] S. Arimoto. *Method of producing a solar cell; a solar cell and a method of producing a semiconductor device*. US Patent 6,093,882. 2000.

[173] "Basic Characteristics and Characterization of Solar Cells". In: *Materials Concepts for Solar Cells*. Chap. 1, pp. 3–43. DOI: 10.1142/9781786344496_0001. eprint: https://www.worldscientific.com/doi/pdf/10.1142/9781786344496_0001. URL: https://www.worldscientific.com/doi/abs/10.1142/9781786344496_0001.

[174] I. Haedrich, U. Eitner, M. Wiese, and H. Wirth. "Unified methodology for determining CTM ratios: Systematic prediction of module power". In: *Solar energy materials and solar cells* 131 (2014), pp. 14–23.

[175] B. J. Pyper and R. M. Peterman. "Comparison of methods to account for autocorrelation in correlation analyses of fish data". In: *Canadian Journal of Fisheries and Aquatic Sciences* 55.9 (1998), pp. 2127–2140.

[176] A. F. Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018).

[177] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[178] A. de Myttenaere, B. Golden, B. Le Grand, and F. Rossi. "Mean Absolute Percentage Error for regression models". In: *Neurocomputing* 192 (2016), 38–48. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.12.114. URL: http://dx.doi.org/10.1016/j.neucom.2015.12.114.