High-Level Simulator of Federation Orchestration

TOON KEYMEULEN

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

High-Level Simulator of Federation Orchestration

Toon Keymeulen
`to3286ke-s@student.lu.se`

Department of Electrical and Information Technology
Lund University

Supervisor: Emma Fitzgerald

Examiner: Christian Nyberg

June 21, 2022

Abstract

The development of a new wireless access infrastructure can introduce some exciting new applications that are impossible to deploy with the current generation communication infrastructure. Therefore the REINDEER project presents a new distributed cell-free network infrastructure called RadioWeaves (RW), that creates favourable propagation and antenna array interaction. As part of the project's solution to a resource allocation problem, this thesis introduces and tests a High-Level Discrete-Event Simulator, enabling the evaluation of various allocation algorithms quickly. In this report, the RW infrastructure will be explained, together with the concept of a utility function and Discrete Event Simulation. Additionally, the structure of the newly implemented simulator will be elaborated, followed by an explanation of the code and the test results.

Acknowledgments

First and foremost, I would like to thank my supervisor Emma Fitzgerald for guiding me through this project. She was always supportive and available to answer my questions. Furthermore, I would like to thank Christian Nyberg for explaining the concept of Discrete Event Simulation and Gilles Callebaut for introducing me to the SimPy framework and assisting me with implementing the WPT and the SIR utility. I am grateful to Liesbet Van der Perre for putting me in touch with Lund University and organising this thesis. Also, I wish to express my gratitude to Ove and his team for suggesting the used channel model and Aleksei Federov for explaining basic concepts in wireless communication. Last but not least, I would like to thank my family and friends for supporting me during this exchange and ensuring my stay in Lund resulted in a fantastic experience.

Popular Science Summary

Development of a Discrete Event Simulator as part of a next-generation communication infrastructure

Imagine being able to inspect real-time processed data from your favourite player in a sports event or receive location-based information in a museum or exposition. Those are two of the thirteen use-cases presented by the REINDEER project, a project that intends to develop and eventually deploy the next generation of wireless communication infrastructure. With the establishment of this new technology, many challenges come into play, one of which is the allocation of antennae to the different connected devices. This thesis introduces a simulator that helps to overcome this challenge.

A major problem within the REINDEER project is deciding which allocation algorithm can be used to coordinate antennae attribution to different devices and applications. As the proposed communication infrastructure will result in a distributed and cell-free network, this decision is not straightforward and needs much attention. One way to quickly evaluate proposed algorithms is by implementing and using them in a High-Level Simulator which leaves out the computationally intensive Ray Tracing but is still capable of representing and simulating a realistic scenario. This thesis introduces a simulator in which small environments can be configured as needed, and simulations can be performed and monitored in time and space. Different applications can be initialized, users can move and appear or disappear, all in the way that is defined in the configuration.

Some major challenges needed to be overcome to develop the simulation mechanism. E.g., how the different entities communicate with each other and how to manage the movements of the user devices in time and free space. This, together with the complexity of developing a model that allows fast allocation of antennas, made this thesis an exciting project.

The REINDEER group can use the simulator to quickly implement and test new allocation algorithms and get a general feeling of how the algorithm will perform in real-life or more detailed simulators. Different kinds of statistics can be recorded to evaluate performance, such as the number of reconfigurations and the amount of insufficiently connected devices. There is also the possibility of getting a general intuition on how changes in the configuration can affect the results. It is possible to analyze some exciting questions, e.g., *"How does the antenna topology influence the frequency of reconfigurations?"* or *"How does the distribution of antennas affect the service provided to a specified application and its needs?"* The simulator's design allows further additions, such as coupling the simulator with a more low-level simulator, after which more realistic communication data can be used to perform allocations.

Table of Contents

List of Figures

List of Tables

List of Acronyms

RW	RadioWeaves
LIS	Large Intelligent Surfaces
RIS	Reconfigurable Intelligent Surfaces
DES	Discrete Event Simulator
MIMO	Multiple Input Multiple Output
LOS	Line Of Sight
SDS	Software Defined Surface
CSP	Contact Service Point
ECSP	Edge Computing Service Point
UE	User Equipment
RW	RadioWeaves
WPT	Wireless Power Transfer
YAML	Yet Another Markup Language or YAML Ain't Markup Language
VR	Virtual Reality
UV	Unmanned Vehicle
AR	Augmented Reality
SINR	Signal to Interference plus Noise Ratio
SIR	Signal to Interference
STU	Simulation Time Unit
LOS	Line-of-sight-

Chapter 1

Introduction

In this Chapter, the role of this thesis within the REINDEER project will be explained (Section 1.1), together with the overall objective of this Master's project (Section 1.2). Previous work and the contributions are discussed in Section 1.3. Section 1.4 concludes this Chapter with an overview and structure of the rest of this report.

1.1 Role Inside REINDEER

This thesis is part of an EU Horizon 2020 project named REINDEER [2]. REINDEER stands for REsilient INteractive applications through hyper Diversity in Energy Efficient RadioWeaves technology. "The REINDEER project will develop a new intelligent connect-compute platform with a capacity that is scalable to quasi-infinite, that offers perceived zero latency and interaction with an extremely high number of embedded devices. It will thereto develop RadioWeaves technology, a new wireless access infrastructure consisting of a fabric of distributed radio, computing, and storage resources that function as a massive, distributed antenna array" [2]. To get a feel for the allocation of resources within the RW infrastructure, a study is proposed to test and implement a first solution. This study consists of two phases, of which this degree project initiates the first phase, creating a high-level simulator that enables a simulation study. The second phase consists of experiments at the KU Leuven test-bed [3].

1.2 Objective

The main objective of this thesis is to develop a High-Level Discrete-Event Simulator which allows the simulation of different allocation algorithms. These algorithms manage the attribution of the different federations used in a specific RW configuration. These so-called federations indicate a grouping of users and the antennas that will serve them. Utility functions are introduced to achieve a fair allocation of antennas to the different federations. Both federations and utility functions will be elaborated in Section 2.5 and 3.5. The results of an allocation algorithm in a certain configuration make it possible to compare its performance and decide which one is more suitable for real-life deployment. The simulator also needs to simulate configurable and dynamic scenarios in which users appear or disappear, move, and change antenna requirements throughout time. This thesis aims not to test different allocation algorithms but to provide a framework that enables this.

1.3 Previous work and contributions

The REINDEER project is an actual state-of-the-art project, and the development of its next-generation communication technology is still in its infancy. That is why no existing simulator exists that implements all the needed aspects to have a realistic simulation of the federation orchestration. There is, however, already software available that integrates similar technology (referred to as Reconfigurable Intelligent Surfaces (RIS)) as RW into a system-level simulator. This RIS can be seen as reflective panels that control the propagation environment of transmitted signals.

To evaluate the potential benefits of deploying RIS into future wireless networks, a study using the Coffee Grinder Simulator is presented in [4]. The presented study aimed to assess the possible performance gains offered by RIS as a function of different physical and technical properties (e.g. deployment density, size, frequency of operation).

Additionally, another simulation software named SimRIS is presented. SimRis is developed to perform simulations of RIS-assisted communication systems. The software can be used for channel modelling of RIS-based communication systems with the possibility to modify many parameters (e.g. operating frequency, terminal and RIS locations, number of RIS elements, environments) [5]. This simulator is designed for passive RIS, which are not used in RW (see Section 2.2.2). It is also only possible to simulate static scenarios in which the allocation of resources is not considered, but the channel model plays a more important role.

In both software projects, low-level channel data and ray-tracing are used. The SimRIS simulator, for example, proposes an initial channel model that can be used for RIS configurations. Unfortunately, these simulators are not tackling the issue this thesis tries to solve; evaluating allocation algorithms in an online setting. Currently existing allocation simulators are covered in Section 3.4.2, once the field of fair allocation has been explored.

As mentioned in the previous Section, this simulator will only provide a high-level simulation of the RW deployment. This means that this thesis aims not to build a Ray-Tracer with the capability to perform accurate calculations regarding communication data and channel models. These calculations would be too much of a burden on the simulator, for which its only goal is to evaluate the coordination algorithms. That is why previously calculated channel estimates are used to obtain a more realistic simulation scenario within the simulator. The researchers within the REINDEER project can quickly implement and test federation coordination algorithms to obtain a general intuition on how they will perform in more low-level simulations and the real world.

1.4 Structure of the report

The next chapter will present a more detailed clarification about the RW technology. In Chapter 3 an overview of the simulator will be given, together with more information about fair allocation and utility functions. The structure and functioning of the code are covered in Chapter 4, followed by the tests and performance measurements in Chapter 5. Additionally, possible future work is highlighted in Chapter 6, completed with a conclusion of this Master's thesis.

Chapter 2

RadioWeaves

In this chapter, the RadioWeaves concept and its challenges and applications will be elaborated on a bit more. On top of that, an explanation will be given about the infrastructure, and the term federation will be introduced.

2.1 Next Generation

The next generation of wireless communication and its possible applications have expressed its need to connect the physical and digital world closer and more sustainably. Current networks cannot implement the needed distributed intelligent networks because of the lack of control of the environment and the increasing power consumption that comes with it [6]. This consumption of previous generation conventional massive Multiple Input Multiple Output (MIMO) has several causes. I.e., installed base stations are not optimized for energy consumption, and setting up new base stations comes with significant complications. Also, the complexity of digital processing, which now accounts for 50% of power consumption, has only increased since the enrollment of the 5G network [6]. At the same time, some newly introduced requirements for 6G are expressed, e.g., very high data rates, low latency, supreme reliability, and great capacity [7]. A new trans-formative wireless concept called RW is introduced to overcome these challenges and create an intelligent radio environment.

2.2 What is "RadioWeaves"

The RadioWeaves technology is based on large-scale intelligent surfaces and cell-free wireless access concepts and will provide favourable propagation and antenna array interaction [8]. Large-scale intelligent surfaces in RW can be seen as a constellation of radioactive panels equipped with antennas. Cell-free networks occur when the antennas are scattered over the coverage area and serve surrounding users. This is in contrast with a cellular network where cell boundaries define what antenna serves which user [9]. The concept of cell-free networks will be further elaborated in Section 2.2.3. The name "RadioWeaves" has its origin in distributed radio and computing power that will be "weaved" into physical buildings and objects. Some of the services mentioned in the use cases and deployment scenarios in REINDEER deliverable [10] have special requirements. Current networks are not able to provide those access and intelligence needs.

The major innovation of the RW concept lies in the proximity of the devices and the radio and computing resources, which offers excellent connectivity and intelligence to the devices. The infrastructure consists of interconnected surfaces with radiating elements, which will dynamically achieve the best performance with the available resources. The probability of being close to specific antennas will be maximized through appropriate placement and orientation of the antennas in the environment. By doing so, each application connected to the RW system will be provided sufficient resources to meet its desired service level.

Furthermore, experiments with massive MIMO [11] have shown that energy still comes predominantly out of one direction, even in reflective environments. RW will try to allocate resources most efficiently, keeping the dominant energy directions in mind.

2.2.1 Distributed antenna arrays and LIS

Several key differences exist between the RW technology and currently deployed networks. The first one is the use of distributed antennas. When we compare with massive MIMO networks, in which cellular base Stations are equipped with a large number of antennas [12], the RW technology will make use of antennas that are spatially distributed around the area. The antennas can be mounted on the walls, integrated into objects, or even 'weaved' into surfaces. The user is then surrounded by radiating interconnected panels. This brings a considerable complexity, e.g., interconnection and coordination become more challenging. The idea is to bring these antennas as close to the user as possible. When a user is close to an area of distributed antennas, beamforming can result in a small region (diameter is half a wavelength) with significantly larger signal power and many other benefits. The interconnected radiating panels are referred to as Large Intelligent Surfaces (LIS), which will be further explained in the following section.

2.2.2 LIS and RIS

In contemporary literature [13][14], large intelligent surfaces are generally considered as reflective and nearly passive surfaces, also known as RIS. Those surfaces can be reconfigured in a certain way to interact with incident signals to improve wireless system performance [15]. In the RW technology, on the other hand, these surfaces will fulfil a more active role and generate signals themselves.

RIS

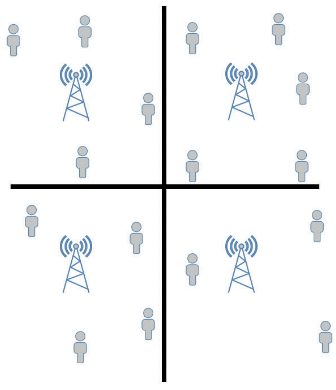
By developing thin films of electromagnetic and re-configurable material, the possibility is created to apply customized transformations to radio waves, thereby avoiding the increasing power consumption and lack of control ability [16]. These surfaces can be described as a two-dimensional array of sub-wavelength scattering particles that transform the waves differently and are called RIS. These two-dimensional arrays are re-configurable by connecting different scattering particles, creating different physical reflection and refraction properties. It is vital to notice the difference between RIS and relays used in MIMO as the primary purpose of RIS is to reconfigure multi-paths in a favourable way for the receiver. The focus of a relay is to re-transmit the received signal, which comes with unwanted self-interference and noise amplification effects. RIS is sometimes referred to as Software Defined Surface (SDS), as the way the surface interacts with the incoming waves is software-defined [17].

LIS in RadioWeaves

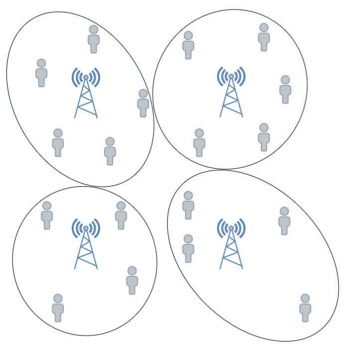
LIS as used in the RW architecture, is distinctively different compared to RIS. The panels which form a LIS are now equipped with a variable amount of antenna elements, which are able to transmit through space actively. A RadioWeaves LIS is physically composed of different elements (sensing element, data storage element, processing element, charging element, radio element, X-haul) [7]. It is demonstrated that the information transfer capabilities of LIS show promising results [18]. For every m^2 of deployed surface, $\frac{\pi}{\lambda^2}$ users can be spatially multiplexed. On top of that, the same study indicated that a small intelligent surface can provide a per-user capacity to around one hundred users in the medium room, virtually as well as if only one user was present.

2.2.3 Cell-Free Networks

Another important difference between this new technology and regular cellular networks is the dense distribution of the different antennas. Usually, a large number of antennas would be centralized in a single base station, and a user device on the same geographical location would always connect to the same station, depending on its connectivity requirements (see Figure 2.1a). With cell-free networks, the tables have turned, and the user is now surrounded by a large number of distributed access points (see Figure 2.1b), which are theoretically able to serve every user at different points in time, depending on the service requirements. In RW, this cell-free concept is taken a step further, as the geographically distributed LIS are now encapsulating the user, and all the antennas are able to provide services to all users at different times. These access points use relatively low power compared to the cellular base stations. These closely located access points open the door to a whole new world of communicational possibilities, accompanied by challenges and risks.



(a) Network with clear distinction of different cells



(b) Cell-free network

Figure 2.1: Figures demonstrating the difference between regular and cell-free networks

2.3 Challenges

Integrating the RW technology comes with several practical challenges. The installation of the infrastructure can cause significant overhead, and integration of the technology might be accompanied by severe aesthetic consequences. As with massive MIMO, the interconnection can lead to bandwidth restrictions, and long connections might lead to significant delays and thereby processing problems. Furthermore, the development of the network is very complex. Because of the different service needs that need to be fulfilled and the many possible allocations, coordinating which antennas serve what users is already a complex problem in a static scenario. It becomes even more complicated when dynamic scenarios have to be considered, users start to move, and requirements start to change.

2.4 Applicatons

The REINDEER project has several major Use-Cases in mind for which the RW technology will intercept the shortcomings of previous network architectures. These bring up many challenges, as they require providing the following aspects [7]:

- wireless energy transfer
- precise position information
- connections to energy-neutral devices
- real-time and real-space interaction within the spatial and temporal reference frame

All these use cases from [19] have their specified technical requirements, so the RW network should be very adaptive and be able to dynamically change the allocation settings depending on the requirements of the applications. Table 2.1 gives an overview of the use cases targeted in the REINDEER project [10].

Use Case	Description
Augmented reality for sport events	The audience of an event, e.g., a sporting match, obtain real-time processed data from the players gathered from sensors that can be seen in AR over the real-time actions.
Real-time digital twins in manufacturing	A digital twin is a full digital representation of that entity, to which data is continuously streamed from its physical counterpart.
Patient monitoring with in-body and wearable sensors	Process and monitoring data derived from in-body and wear-able sensors offers the opportunity to provide more efficient, better, and possibly remote healthcare.
Human and robot co-working	In future factories and care environments, humans and robots will need to cooperate to perform tasks or for robots to provide care to patients.
Tracking of goods and real-time inventory	Tracking of goods allows for a holistic vision of the supply chain and inventory status, enabling real-time decisions based on the information collected from each single item present.
Electronic labelling	Electronic shelf labels display dynamic information. Through WPT, RadioWeaves enables the use of energy-neutral electronic labels, with display information updated wirelessly and inventory located and tracked with high precision.
Augmented reality for professional applications	AR glasses need high-resolution displays and cameras and high-performance image processing while requiring lightweight and ultra-low-power designs for wearability and usability.
Wander detection and patient finding	Through patient finding and wander detection and prevention, patients, e.g., with dementia, get more freedom of movement in combination with a lower risk of accidents.
Contact tracing and people tracking in large venues	Contact tracing and people tracking could enable companies or events to log unsafe contacts or give insights for safety measures in case of, e.g., the Covid-19 outbreak.
Position tracking of robots and UVs	The positioning of robots and unmanned vehicles can improve autonomous driveability and productivity and reduce the loss of assets.
Location-based information transfer	Active information can be given depending on the user's location.
Virtual reality home gaming	VR is a growing trend within video gaming, including in private homes on consoles and personal computers with a connected headset.
Smart home automation	Relatively dense arrays of energy-neutral sensors could be deployed throughout the home to provide a three-dimensional map of environmental quantities, thus allowing smarter home automation.

Table 2.1: Use cases with their description

2.5 Structure and Federations

This section briefly overviews the components that make up the RW architecture and their functions. Secondly, the earlier concept of federations is explained in more detail. Further information about RW terminology and structure can be found in [6][7].

2.5.1 Components

In Figure 2.2 an overview is given of the different network components present in a typical RW setup [6]. We can see different Contact Service Points (CSPs) which are responsible for WPT, sensing, and communicating with the User Equipments (UEs) or devices on which different applications are running. These CSPs are the equivalent of the antenna array, which, when placed together on a surface, form a LIS. The Edge Computing Service Points (ECSPs) mentioned in the figure serve as processing units that are spatially distributed over the area. Their primary purpose is to offload processing power from the CSPs and coordinate and aggregate data to and from the backhaul. Both CSP and ECSP are terms made up for the REINDEER project. Cooperation of all the different CSPs will also require a certain degree of synchronization for which synchronization anchors are introduced [6]. In this thesis, only the CSPs and UEs are considered. By doing this, the system is a lot more manageable, and the problem of allocating resources within RW is relaxed.

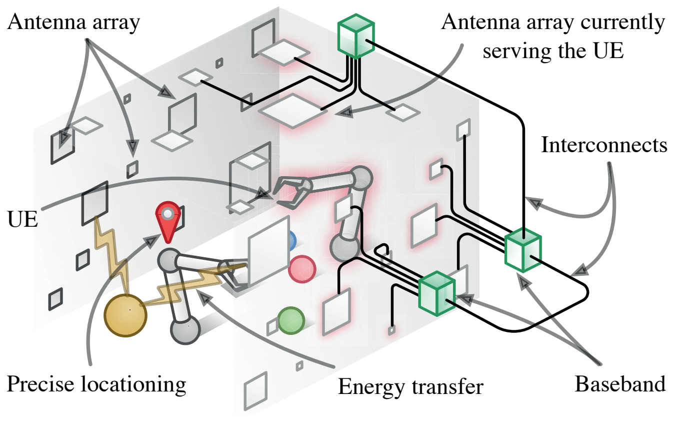


Figure 2.2: Overview of RW infrastructure, created by William Tärneberg

2.5.2 Federations

As there might be a lot of different UEs, not all the resources in the system will be able to contribute to the same service required by the applications on the user devices. Therefore **federations** are introduced. As mentioned in [7]; "Dynamic federations consist of constellations of antennas, edge computing units, data storage, and other resources, to serve a specific application". As the objective of this thesis is the allocation of CSPs, a federation in this project is considered as only a group of UEs and CSPs that are interconnected with each other to provide the communication requirements. These federations can contain CSPs that are positioned close to each other, but that is not a necessity, as some applications might have other requirements, i.e., spatially distributed for accurate positioning [6]. Depending on these requirements of the applications, different CSPs will be assigned to the different federations.

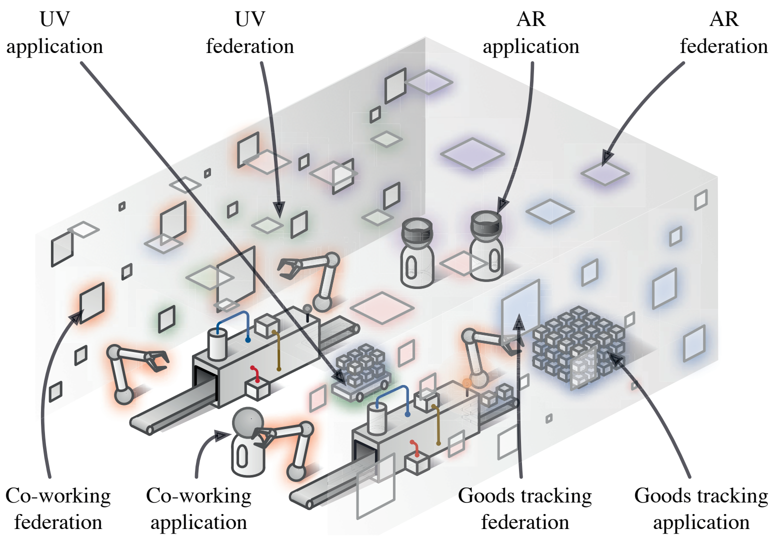


Figure 2.3: Figure of some use cases, created by William Tärneberg

Figure 2.3 shows four use cases mentioned in Table 2.1. Each of these use cases is attributed a colour. The colour of the rectangular surfaces on the wall indicates what application and federation they belong to. Because certain panels might be desired by more than one federation, a mechanism is needed to manage this difficulty. This is where allocation algorithms come into play. These algorithms will each have their way of attributing panels to federations. It is the goal of this thesis to enable the evaluation of each allocation algorithm.

Chapter 3

The Simulator

This chapter elaborates more on the different elements used to implement the simulator. Section 3.1 and 3.6 give a brief recapitulation of what the explicit requirements are for the project and how they will be fulfilled. Additionally, Section 3.2 provides an overall elaboration on simulation, as well as the necessary concepts needed to understand the underlying base behind Discrete Event Simulation. Fair allocation and utility functions also play a vital role in this Master's thesis and are discussed in the final Sections 3.4 and 3.5.

3.1 Requirements

The overall objective of this thesis is to create a simulator that enables to quickly implement different federation coordination algorithms and compare the test results. Even though this remains a high-level simulator, several important aspects need to be included in the program. First, there must be the opportunity to configure a specific physical area and its associated CSPs. Furthermore, the different users who need to be served by the different federations should be able to run different applications with specific needs. These users should also be able to move through the physical area. Another major part of the project is the actual allocation of the CSPs and the ability to express a preference between them. This should be done by using preference values configured in the configuration files.

3.2 Simulation

Simulation can be used as an analysis tool for predicting the effect of changes to existing systems and a tool to predict the behaviour of new systems in diverse configurations. In our case, the simulator is developed to analyse the behaviour of different allocation algorithms in possible future circumstances.

3.2.1 The system

To perform any simulation, it is required to define a **system** and a **system boundary**. "A system is defined as a group of objects joined together in some regular interaction or interdependence toward accomplishing some purpose" [20]. In our case, the system generally consists of UEs and CSPs, which interact with each other to let the different applications run their service. Changes outside the system (**system environment**) can sometimes affect the system. However, in the case of this project, the influence of these changes can be neglected, and the **system boundary** can be clearly defined as everything that happens outside of the configured environment.

Within the defined system, several important components can be defined:

- An **attribute** is a property of an entity.
- An **activity** represents a period of a specified length.
- The **state** of a system is defined as the collection of variables necessary to describe the system at any time.
- An **event** is defined as an instantaneous occurrence that might change the system's state.

3.2.2 The model

To study a system, a **model** must be defined. This model represents the system and only includes aspects that affect the problem under investigation. A simplification of the system is required to prevent the model from being unnecessarily big. However, it is necessary to remember that the model needs to be sufficiently detailed to draw any valid conclusion about the represented system [20].

Our model consists of a physical environment defined by three dimensions, as shown in Figure 3.1. Within this area, three different **entities** can be defined; **UEs**, **CSPs** and physical **objects**. These entities have a high number of attributes; id, position (UE and CSP), direction(UE), ... The activities in the model mainly consist of movements of the UEs and the sending of messages. As the name of this thesis suggests, a discrete **event** simulator mainly focuses on the occurrence of events. These events are explained in the following section.

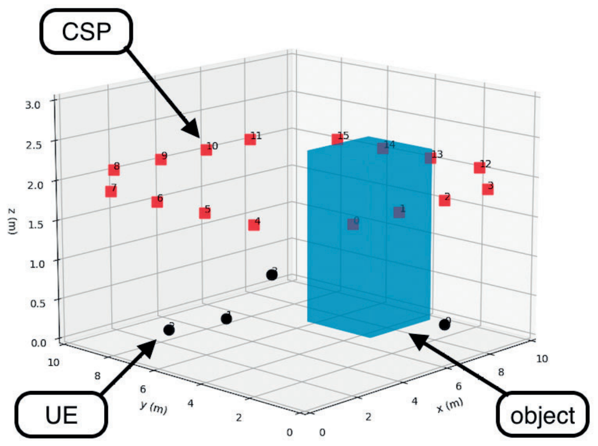


Figure 3.1: Illustration of model in the simulator

3.3 Discrete Event Simulation

Discrete event simulation distinguishes itself from other simulations by the fact that state variables only change in a discrete set of points in time [20], which is called **next-event time progression**. Thereby the system is in a static state during the period between two consecutive events. The simulator enables the different entities that make part of the system to choose the next time it will change its state or that of the system. A clock is mainly used to keep track of current simulation time and is only incremented by 'hops' to the time the next event occurs. This way of handling time in a simulation makes Discrete Event Simulator (DES) in most cases faster than simulators using **fixed-increment time progression**. Because with the fixed increments, the system has more points in time where state changes need to be evaluated. The earlier mentioned clock makes use of a unit called Simulation Time Units (STUs). What this unit represents in the real world depends entirely on what is being simulated. According to the system, the STU should be interpreted differently, e.g., the simulation of a solar system will probably have a different STU (e.g., years) than a program simulating crowd control (e.g., seconds). In the case of this project, the STU of choice is seconds. Most of the UE movements inside the configuration will move at a speed that can most easily be presented by $\frac{m}{s}$, and larger or smaller scaled time units seem less convenient.

In opposition to DES, there is also continuous simulation in which systems are analysed by analytical methods rather than the numerical method. Although the system we are simulating is relatively more continuous than discrete because of the continuous activities (f.i. the movements of the users in the environment), we still decide that because of the study's objective, the use of a discrete simulation model is more appropriate. The use of DES is sufficient as there is no need to obtain very precise channel data, and the speed of DES ensures fast results.

3.4 Fair Allocation

An essential step in the process of federation orchestration is deciding how the different CSPs will be assigned to the different federations. We seek to do this by using fair allocation algorithms. Nevertheless, because different sectors have interested in good allocation algorithms, being fair in terms of allocation can mean many different things. This section tries to shed light on what fair allocation entails.

3.4.1 What is Fair Allocation

Fair allocation refers to the general problem of fairly dividing a shared resource among agents having different and sometimes antagonistic interests in the resource [21]. To clarify this, let us take the example of distributing candy to many kids. In this setting, the kids are the agents, and the shared resource is the collection of different candies. As you are a good person, it makes sense to distribute the candy as fairly as possible among the kids who might have different preferences. If we assume the pieces of candy can not be divided into smaller pieces without losing value, it is not hard to imagine this task can become quite hard.

The first distinction we have to make before diving deeper into the world of allocating resources is between divisible and indivisible goods. The name indivisible speaks for itself; these goods lose value when broken down into smaller parts. They are classified as indivisible (f.i. a car or a bike). On the contrary, divisible goods can always be divided into smaller parts without losing value (f.i. a cake, a field, and money). The division of indivisible goods is often more difficult because certain fairness concepts like **envy-freeness** or **proportionality** can be unreachable (see Section 3.5.1). As CSPs can not be subdivided into smaller elements, we consider the resources in the RW concept as indivisible goods. We can present these goods as a set of objects $O = \{o_1, o_2, ..., o_n\}$ (indivisible). In the RadioWeaves infrastructure, CSPs are seen as shareable. They can be part of a part of another federation at a different point in time. With the correct scheduling mechanisms, CSPs might even be part of different federations concurrently. The CSP would then serve each federation a certain percentile of its time. However, in this thesis, CSPs are only seen as non-shareable. They can only be part of one federation at a time, but this federation can change over time.

As in the case of distributing candy, each agent can have different **preferences** regarding the available goods. This feature of fair allocation can lead to a lot of added complexity. As comparing the number of available shares is huge most of the time, the explicit representation of the agents' preferences becomes unrealistic.

From now on, the notation of [21] will be used. $N = \{1, 2, ..., n\}$ represents a set of agents. Apart from allocating individual objects it's also possible to allocate a bundle $s = \{o_1, o_1, o_3\}$ which represents a subset of O. An *Allocation* is a function

$\pi : N \rightarrow 2^O$, mapping each agent to the bundle it receives, such that $\pi(i) \cap \pi(j) = \emptyset$ when $i \neq j$. This means that no goods can be part of bundles allocated to more than one agent; the items can thus not be shared. An allocation is *complete* when $\cup_{i \in N} \pi(i) = O$ and the set of allocations or total allocation is denoted Π. The utility of an allocation is defined as u_Π

Agents' preferences for goods can be defined differently, being ordinal and by a utility function. In the ordinal case, agents must be able to express an ordinal preference between the different available goods and bundles. Agents must be able to rank all the items, with not no items being equal. This thesis only focuses on the *utility function* $u_i : O \rightarrow F$ in which each object can be mapped to a score. As different agents can have different preferences we can present the set of utility functions as $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. To demonstrate the complexity of fair allocation, some terms used in the context of fair allocation are introduced [22][21].

- **Pareto efficient:** A total allocation is Pareto efficient if no new total allocation can be made in which the total utility of an agent rises without decreasing the total utility of another.

$$\Pi_x \text{ is pareto efficient if } \nexists\ \Pi_y : u_i(\pi_{\Pi_y}(i)) > u_i(\pi_{\Pi_x}(i)) \text{ and}$$
$$u_j(\pi_{\Pi_y}(j)) \geq u_j(\pi_{\Pi_x}(j)) \text{ for every } i, j \in N$$

- **Maxmin allocations:** A set of allocations is **maxmin** when the utility of the poorest agent is maximised [21]. This does not necessarily mean that the allocation is also Pareto efficient. A MaxMin allocation only focuses on maximising the utility of the poorest agent and does not consider the allocations of the other agents.

$$max_{\pi \in \Pi} \left\{ min_{i \in \mathcal{U}} u_i(\pi(i)) \right\}$$

- **Envy-freeness:** This criterion of fairness is full-filled when no agent feels envy for any other agent. It means that when the resources are allocated to the agent, every agent should prefer its partition over the allocated partitions of the other agents.

$$u_i(\pi(i)) \geq u_i(\pi(j)) \text{ for all agents i,j} \in N$$

- **Equitable:** When a set of allocations is equitable it means that every agent that is considered has the same utility.

$$u_i(\pi(i)) = u_j(\pi(j)) \text{ for all agents i,j} \in N$$

- **proportionality:** When proportionality is used as a Fairness criterion, each agent should receive a bundle of goods that they value at least as much as 1/n of the entire allocation. Because the allocation of CSPs in our setting is an allocation problem of indivisible goods, the property of proportionality is not guaranteed.

The existence of all these different notions of fairness emphasises the need for a simulator that can evaluate algorithms. Depending on what notion of fairness is the goal to pursue, a different allocation algorithm might be suitable.

3.4.2 Online vs. Offline

When allocating resources, it is vital to notice the difference between online and offline situations. Agents are referred to as being online when their preferences or presence varies over time. This is the case with RadioWeaves, as agents can appear or disappear. Resources are online when the availability is not fixed but can vary over time. The main difference between online and offline situations is the assumption that all the items are available at any time in offline situations, which does not hold for online situations.

Looking at the difference between online and offline situations, from now on, we primarily consider the online variants as these seem to be most relevant regarding allocating resources in the RW system. We can consider the different applications and federations that serve them being the agents of the system and the panels being the goods that need to be divided by the allocation algorithm.

We notice that agents in our deployment can change their requirements over time. They will also move in the environment, which changes the different utility functions. That will be explained in the next section. As these changes are dynamic and happen over time, we should also consider allocation algorithms in a dynamic setting.

In [23] two online allocation algorithms (OnlineMaxDelivered, OnlineMaxSatisfied) are presented, which take into account online agents. Agents can arrive and depart, and the allocation of the resources is done without the knowledge of future intervals. The downside of the allocation algorithm presented in this work is that the allocation is performed based on requirements, and no valuation functions come into play. This conflicts with our setting, which will be further elaborated in section 3.5. Another big difference between both settings is the type of resources used in the allocations. In [23] the resources are perishable. Perishable is defined as not storable, and if not used directly, it is wasted. In our configuration, the goods (CSPs) are considered permanent.

Another allocation procedure presented in [24] supposes additive utilities and full independence agents, another two properties which are not applicable in our setting. It is demonstrated in this paper that the expected utilitarian social welfare is maximised when agents take alternate turns in picking items. Utilitarian welfare exists when the sum of all agents' utilities is maximised. This paper does not take into account online agents, another reason why this allocation is not applicable to our setting.

In [25] the "HOPE-ONLINE" algorithm is presented. Although the guarantee of Pareto-efficiency, envy-freeness and proportionality seems to be an impossible task, this algorithm guarantees approximately-fair allocations utilising distributional knowledge of the system. Agents arrive sequentially, with unknown stochastic utilities. By accessing historical data, equitable access to a resource is strived for. Using previously obtained data might be interesting to investigate in the case of RadioWeaves. E.g., Large and similar events might always have the same arrival distribution of users. An algorithm that makes use of this knowledge might produce better results.

3.4.3 Information

[26] introduces another dimension of fair division. Additional to the online vs offline situation, there is the informed vs uninformed setting. The system has information about the items or agents yet to arrive in the informed setting. The system presented in this thesis is focused on the algorithms which perform allocations from the uninformed perspective, i.e. no information about the future presence of UEs or movements is available. Algorithms that rely on full information are thus no longer applicable in this scenario.

Guaranteeing Pareto-efficiency, envy-freeness, and proportionality simultaneously seems impossible in the uninformed setting; thus, the challenge is defining meaningful notions of approximately-fair online allocations and developing algorithms that utilise distributional knowledge to achieve such allocations.

3.4.4 Research

A reasonable amount of research has already been conducted on fair division. In [27] Procaccia and Wang show that MaxMin allocations might not exist with additive valuation functions. On the other hand, they prove that allocations always exist where agents receive $\frac{2}{3}$ of the valuation of the least desirable bundle (if all items are divided between agents). This result is later improved [28] to a $\frac{3}{4}$ factor in the additive setting. Their results are also extended to some non-additive settings.

Additionally, [29] reaffirms that envy-free allocations are not always possible in settings with indivisible items. In that case, the optimisation problem of achieving minimum envy introduces itself. In the presented work, monotone, additive utility functions are associated with each agent, and an algorithm is presented to minimise the envy-free ratio.

In [30] Asadpour and Saberi present an approximation algorithm for Max-Min fair allocation of indivisible goods. The algorithm assumes known linear utility functions, and achieves Max-Min utility with an approximation ratio of $\Omega\left(\frac{1}{\sqrt{k}log^3k}\right)$. The algorithm first guesses the optimal solution using a binary search tree. Subsequently, matches between goods and agents are made, after which conflict resolution is used to allocate goods assigned to more than one person. A general overview of fair allocation can be found in [26].

We can conclude that the world of fair allocation is an extensive and complicated one. A wide variety of different settings make it very interesting to research. On top of that, the different notions of fairness to strive for make it hard to generalise this field of interest and conclude an optimum algorithm. The introduction of the RW brings another new setting that needs to be studied.

3.5 Utility Functions

When comparing agents' preferences to bundles, one could say the utility of a certain bundle can be determined by just adding the different utilities of objects in that bundle, items are then unrelated, and this we call *additive preferences*. However, this is not always the case, e.g. when objects of similar nature are closely coupled. Imagine o_1 being a plane ticket to Paris and o_2 a plane ticket to New York. Both these objects can be valued by an agent in terms of a certain utility, but if these flights happen to be on the same day, the *additive* assumption is not valid anymore. That is because there exists a certain dependency between both objects. In the case of federation orchestration the utility functions also express utilities and can be *subjective*, *complementary* and *substitutable*[31].

Subjective: "Different sets of UEs within a federation can have different utility values for different CSPs or constellations of CSPs"

Complementary: "A group of CSPs can have synergistic value, e.g. multiple CSPs close to each other (in terms of the CSP topology) for low latency"

Substitutable: "Having one of a group of CSPs has value, but more than one is not needed, e.g. CSP located close to each other may give poor spatial diversity for positioning"

The attribution of a *utility* or value of a *bundle* of CSPs to a federation is done by a certain *combinatorial* utility function. This function varies depending on the applications, and the requirements the federation and UEs are serving. We can distinguish several main performance metrics that focus on different wireless communication aspects. Even though these metrics focus on a different aspect of wireless communication, applications will likely have a utility function that combines these performance indicators. The performance metrics used to obtain utility values are mentioned below, with some of the use cases (see 2.1) that value these metrics the most.

Data rate: Virtual Reality (VR) home gaming, location-based information transfer, ...

Latency: Augmented Reality (AR) for sport events, position tracking of robots and Unmanned Vehicle (UV)s, ...

Positioning accuracy: Real-time digital twins in manufacturing, tracking of goods and real-time inventory, ...

Received power for WPT: Smart home automation, AR for professional applications, ...

Packet Error Rate: Real-time digital twins in manufacturing, Human and robot co-working,...

3.5.1 Utility Levels

There are several levels on which the utility of an allocation can be evaluated in this project, and each level has some good reasons to be taken into account. The lowest level is that of the users themselves. For each user, there is a normalised utility value that expresses how satisfied the user is with the allocation of its appointed panels. There can be a large variety in how certain allocations express themself in a normalised utility value. As some users might not be more satisfied when more panels are added to their federation, and some others might only have a utility value different from 0 when a certain threshold has been met.

The second level of evaluation is the utility of a single federation. We can obtain this utility in a few different ways; depending on what applications are run within the federation, a different way of evaluating the utilities might be more significant. Applications where all users should have secure performance, will be better off by evaluating the minimum utility of all its users. In comparison, applications with more flexible performance thresholds can be evaluated fairer by looking at the mean of the utilities within the federations. Last but not least, we need a way of expressing how good a total allocation is compared to others. This can be done by combining the utilities of the different federations.

3.6 The idea

Different significant components will be needed in the simulator to fulfil the requirements mentioned in Section in 3.1.

A first part of the simulator would consist of all the different entities needed to represent an actual RW deployment. Classes like CSP, UE and federation seem indispensable. Another crucial component will be a class that can calculate the utility value for a certain UE, federation or deployment.

Additionally, a component will be needed to visualise the simulated deployment. Preferably this visualisation will not only be able to show allocations but also have a dynamic debugging function. This debugging mode would let the user step through time to see what is happening clearly. This visualisation will need to be done in 3D to maintain a clear view of how the UEs are moving.

The last component is responsible for the actual allocation of the CSPs The allocation of a particular CSP to a federation will be done in two main steps. In the first step, each federation calculates a certain utility value for each service point. This value expresses how much the federation 'wants' this service point to be part of its federation regarding the service requirements of the application it serves. In the next step, the allocation algorithm will decide what service points will be distributed to the different federations. As mentioned earlier, a motivation for this Master's Thesis is to provide a platform to test these different allocation algorithms and compare the results to see which way of allocating resources is more favourable.

Chapter 4

The Code

This chapter will look at the different aspects of the source code that make up the simulator. First of all, Section 4.1 gives an overview of the project's structure. This overview can help in understanding the approach used to build the simulator. This section is followed by an elaborate explanation of how the different utilities are determined in the project. Afterwards, Section 4.3 introduces the SimPy framework, and Section 4.4 discusses the different SimPy processes that are implemented. Section 4.5 illustrates the integration of objects in the simulated environment. Finally, the total simulation mechanism will become clear with a virtual simulation run in Section 4.6.

4.1 Structure

The project is organised to make the code as structured as possible regarding readability, reusability, and modularity. Therefore specific packages are created so that each group of different files have components with a similar role. In the following sections the most important packages and classes are discussed.

4.1.1 Entities

The entities used in the code are presented in this section. The following explanation will briefly cover the most important attributes and methods.

- **allocation:** This class does not contain anything more than a list of federations. It is introduced to encapsulate a full allocation. It is easy to add attributes to this object that can give interesting insights, for example, the total number of ignored reconfiguration messages sent within the allocation.
- **application:** The application class is nothing more than an object which holds much information used by the UEs and federations. E.g. Threshold levels, utility valuations, the maximum number of users etc. This class does not contain any operative code.

- **federation:** A federation object holds a set of UE objects and a set of CSP objects, together with an application object. On top of that, each federation controls its events responsible for serving the **federation process** that will be elaborated in Section 4.4.2.

- **CSP** and **UE:** The CSP class is a small class that holds its position in the physical environment. The UE class, on the other hand, is a bit more complicated. Just like the federation, it also holds a process (see Section 4.4.1). In addition, some attributes are stored from the configuration like the speed of the UE, the time to appear and disappear, etc.

- **ray**, **object** and **polygon:** These three classes are introduced to perform the ray tracing briefly explained in Section 4.5. If the ray tracing option is configured to be "false", these two three are not used in the simulation.

- **reallocation entity:** The reallocation entity is the last class that contains SimPy processes; the reallocation process (see Section 4.4.3) and a Logging process that makes use of the Logger class mentioned in Section 4.1.5. This entity initiates the reconfigurations through time and holds the allocator object configured to hold the preferred allocation algorithm class.

- **Vector** and **Waypoint:** The last two entities are essential for the UE movements. The calculations of speed and direction are based on the vector class, and the trajectory of an UE is described by a list of waypoints. These waypoints also keep track of the time the user should spend at that particular waypoint.

4.1.2 Configuration

This package speaks for itself; it contains all the different configuration files which make op a particular RW configuration. The different CSPs and UEs can be defined here, together with some application files which give more information about what utilities should be used and how the utility should be evaluated on the different levels. Examples of these configuration files can be found in the Appendix(A.2). Each mentioned entity is defined in a separate file to obtain a more readable configuration. In the configuration package, we can also find a script that can generate the different configuration files based on what type of configuration needs to be simulated. This script was created by Emma Fitzgerald to be used in a prototype and was later modified by myself to be compatible with the final version of the simulator. New configurations can be set up by modifying some parameters or defining a new generator function in the script. There is, for example, a method that is able to deploy CSPs in an evenly spaced way or a function that creates a trajectory for a user based on given parameters and objects in the simulated environment. The language used for the configuration files is Yet Another Markup Language or YAML Ain't Markup Language (YAML) and will be elaborated on in the next section.

YAML

YAML is a compact form of XML, and it is used for defining the different configuration files. It is carefully selected because of different criteria. As the configuration of a RW deployment can become quite large and complicated, human readability was an important selection criterion, as well as its flexibility and accessibility. On top of that, YAML allows the user to use comments in the files, enabling in-file documentation and the declaration of the different options. A downside of this language is the absence of support for some programming languages. Fortunately, Python has an additional library that can be used to parse these files.

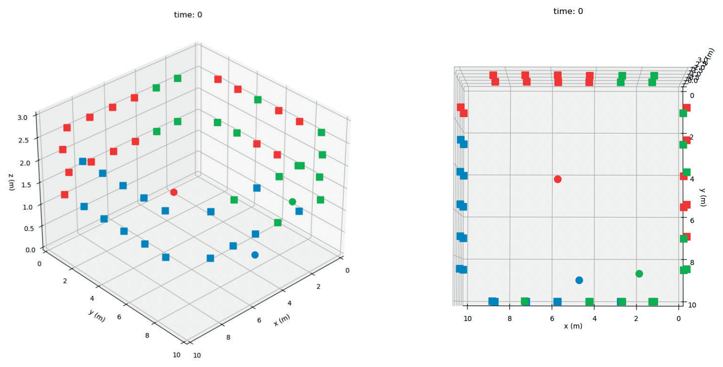
4.1.3 Allocation

All the code related to the actual allocation of the CSPs to the federation is grouped in the **allocation** package. The different currently implemented allocation algorithms are stored in this package, together with an interface that can be used to implement future coordination algorithms. Some currently existing simulator algorithms are the ordinary **round_robin** algorithm and a round-robin allocation where a particular allocation pattern can be passed to. Besides that, there is also the **shrinking algorithm** which is again implemented by Emma Fitzgerald. To compare the performance of allocation algorithms in terms of maximising the utility of an allocation, there is also a more heuristic approach to solving the coordination problem. The metaheuristic **Simulated Annealing** can be used to benchmark the results of newly implemented allocation algorithms. Additionally, a **Steepest Descent** approach is available to locally optimise a given configuration. Ultimately there is also a **Random Allocation** algorithm that is used to test the performance of the simulator without the computationally intensive shrinking algorithm. It can also be used as a lower bound for any other algorithm, as the algorithm's performance should be in between that of the random allocation and an optimum. Future research should determine if such an optimum may or may not be possible to find.

Different allocation algorithms can be added to the simulator as long as they are compatible with the **allocation interface**. A significant burden when defining algorithms in this setting is the strange behaviour that can be unique to each defined application. As mentioned in Section 3.5, this behaviour can consist of utilities dropping to 0 when a certain threshold is not met or utilities not being linear according to their correlated performance metric. A typical approach to the allocation would be to start from a minimal assignment where all the federations have one or no CSP assigned to them and then iteratively add CSPs to the federation according to the added utility to that federation. Unfortunately, this approach would not work in our case as the threshold requirements might not result in higher utilities for a single CSP addition case. This could result in certain federations suffering from starvation and not getting any new CSPs assigned to them. To still perform meaningful allocations in the simulator, the shrinking algorithm is used in the simulator. It was created by Emma Fitzgerald and Gilles Callebaut and will be presented in [31].

4.1.4 Visualisation

Another package is introduced to decouple the calculations and visualisation of the deployment. This package contains a visualiser object whose only responsibility is to convert a given allocation into a 3D representation of the environment in which the different federations can be visible. To do so, new entities as **csp_3D** and **ue_3D** have been introduced to keep the code for visualisation as separated as possible from the rest.



(a) Angled view 3D visualisation **(b)** Top view 3D visualisation

Figure 4.1: Examples of 3D visualization

Figure 4.1a and Figure 4.1b shows an example of a simulator visualisation from different angles. We can distinguish the different federations by colour. Different symbols have been used to distinguish the CSPs and UEs from each other. The CSPs are represented by squares, and the UEs are represented by dots. This way of visualising has some significant advantages, with the first being the ability to visualise the model at runtime. By switching back and forth between the UI loop and loop of code, debugging of the code can be made a lot easier, as well as having more transparent insight into how the model changes state over time. The next event is taken from the event loop and processed by the simulator by pressing any key on the keyboard. The effects can be shown directly at runtime. On top of the visualisation, the time is shown to track what time the simulation is currently at.

The disadvantage of using this way of animation is that it is not very appealing to the eye. The distinction between CSPs and UEs is sometimes hard to make, and the capabilities of the matplotlib software to show realistic 3D environments are limited as it is not the original purpose of the framework. Except for the fact that the dimensions of the 3D graph can be changed, there is little flexibility: no built-in 3D bodies, no textures available, and no complex environments.

4.1.5 Utils

This package contains three elementary classes; the Parser, the Oracle and the Logger. The Parser uses the YAML library mentioned earlier in Section 4.1.2.

The Oracle is probably one of the essential classes included in the simulator and has a vital role in the functioning of the simulator. Additionally, there is also a Logger class which can be used to log valuable data to CSV files. This data can be analysed to gain insights (f.i. utility over time).

Oracle

The oracle class is the class responsible for calculating all the different utilities used to make the allocations by the *federation allocator*. This class also contains the methods to evaluate the utilities on the three different levels as mentioned in Section 3.5.1. To evaluate the utility of a user within a federation, the different utilities (WPT, data rate, packet error rate, latency, and positioning) are calculated and combined, taking into consideration the values of the utilities determined by the application that is running on the UE. By combining all the utility values of the users within a federation, a new value can be determined that expresses the total utility of that federation. How these values are combined into a federation utility also depends on what application is hosted by the users. At last, there is also a method that can calculate the total utility of an allocation by joining the federation values in a certain way.

The way that the utility calculations are implemented is very adaptive. It requires minimal effort to change how, e.g., the WPT value is calculated as it would only require changing the implementation of that specific method without bothering the rest of the simulator. On top of that, it is essential to keep in mind that the following utility functions do not determine the utilities by taking into account real physical metrics. The methods to become a utility value are supposed to be indicative and are designed to follow the same characteristics as real-world behaviour. Following is a brief overview of how the different utilities are calculated.

4.2 Utility Calculations

4.2.1 Wireless Power Transfer

This utility is being measured by the energy of the signal the user receives in its federation. Ove Edfors and his research group at Lund University suggested the channel used to obtain the different signal data. These calculations are based on the formula explained in [32] and given in Equation 4.1.

$$g = \sqrt{\beta} * h \tag{4.1}$$

In the formula above, the positive real number β represents the *large-scale fading* coefficient. This coefficient embodies range-dependent path-loss and shadow fading. The *small scale* fading is represented by h and is a complex number that makes up for constructive and destructive interference. Small-scale fading is assumed in the simulator to be Rayleigh; that is $h \sim CN(0, 1)$. This model is chosen as it is widely used in the existing literature on wireless channel modelling, including on massive MIMO. Additionally, it is tractable to use in theoretical analysis, allowing results obtained by the simulator to be compared to analytical modelling. At last, more realistic channel models are currently being worked on. This channel model will then be connected to the simulator in the future. Therefore it is not worth implementing another specific and complicated channel model. The noise can be configured in the configuration files, and the exact calculations to obtain the transferred power are given below.

Symbol	**Description**
$d_{k,l}$	Distance between UE k and CSP l
λ	Wavelength in meter
K	Number of UEs in that federation
L	Number of CSPs in that federation
N	Number of configured antennas per CSP
$\beta_{k,l}$	Path loss for user k and CSP l
$\mathbf{h}_{k,l}$	Channel vector for all antennas at CSP l for UE k
$\mathbf{w}_{k,l}$	Precoding vector for all antennas at CSP l for UE k
$r_{k,l}$	Total received signal for for UE k and CSP
$CN(0,1)$	Rayleigh small scale fading
$n_{k,l}$	Noise in UE k from CSP l

Table 4.1: Signal and interference notations

$$\mathbf{h_{k,l}} = \sqrt{\beta_{k,l}} CN(0,1) \tag{4.2}$$

$$\mathbf{w}_{k,l} = \frac{\mathbf{h}^*_{k,l}}{||\mathbf{h}_{k,l}||} \tag{4.3}$$

$$P_{k,l} = \frac{P_{tot,l}}{K} \tag{4.4}$$

$$r_{k,l} = \sqrt{P_{k,l}} w_{k,l} s_k h_{k,l} + n_{k,l} + \sum_{k' \neq k}^{K} \sqrt{P_{k',l}} w_{k',l} s_{k'} h_{k,l} \tag{4.5}$$

$$r_k = \sum^{L} r_{k,l} \tag{4.6}$$

$$P_{dB} = 10 \times log_{10} \left(\frac{|r_k|^2}{P_{ref}} \right) \tag{4.7}$$

The path loss values $\beta_{k,l}$ are obtained by taking the configured wavelength and the distance between the UE and CSP. We consider only antennas that are isotropic and have no directivity. Both distance and wavelength, together with Equation 4.8 then give the free space path loss.

$$\beta_{k,l} = \left(\frac{\lambda}{4 * \pi * d_{k,l}}\right)^2 \tag{4.8}$$

The resulting value P_{dB} from Equation 4.7 can then be normalised to a value in the [0-1] interval by an equation adapted to the used configuration. Equation 4.9 shows such an equation. *max_power* is the approximated maximum power value a UE can obtain in a specific configuration. This value can be determined by assigning all the CSPs in the configuration to the same federation as the UE. By placing the UE close to a wall that is very densely covered by CSPs, an estimation can be made of what the maximum power value should look like. *user_power* represents the signal power of the UE of which the *WPT_utility* is being determined.

$$WPT\_utility = \frac{1}{max\_power} \times user\_power. \tag{4.9}$$

4.2.2 Positioning

To obtain a positioning utility, a way to express how well an entity is surrounded by CSPs must be introduced. One way to measure this is to look at the angles the user makes with the different CSPs included in its federation. These angles are defined in a spherical coordinate system shown in Figure 4.2. The polar angle, or the angle from the vector with the z-axis, is denoted as θ. The other angle ϕ is called the azimuth angle. The simulator calculates a utility value by obtaining the azimuth and polar angles between the UE position and all the CSPs contained in the federation. Subsequently, the biggest gap between all azimuth angles (respectively polar angles) is calculated and mapped to a corresponding utility value in the [0-1] interval. The interval for calculating the biggest angle gap is [0-360°] for the azimuth angle and [0-180°] for the polar angle.

To clarify, we take a look at an example. Let us say the azimuth angles a UE makes with four different CSPs are [20°, 40°, 80°, 240°]. The corresponding angle gaps with the next angle in line become [20°, 40°, 160°, 140°]—the last angle results from subtracting 20°(+360° interval) - 240°. The biggest value(160°) is subsequently converted into a utility value in the [0-1] interval. The bigger the biggest angle gap, the smaller the utility value. This can be explained by the fact that a bigger angle gap indicates a larger spatial area uncovered by a CSPs. Similar calculations for the polar angle with the [0-180°] interval can be obtained.

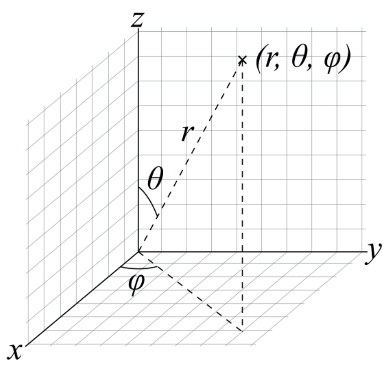


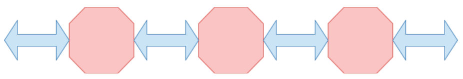
Figure 4.2: Spherical coordinate system [1]

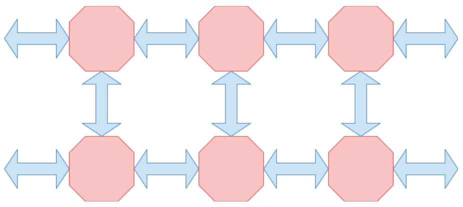
4.2.3 Latency

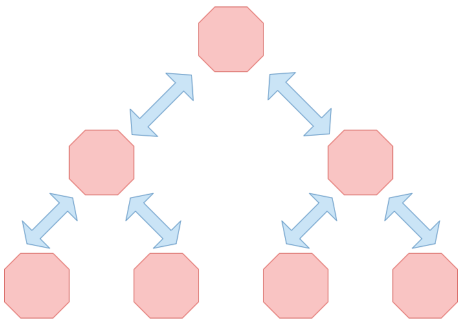
Latency is calculated by taking into account the topological distances between the different CSPs concluded in the federation. Distances are expressed in the number of hops in the topology. These CSPs can be interconnected in various ways, of which three of these are implemented in the simulator and given below. How these CSPs are interconnected greatly influences the amount of hops messages have to take to reach their destination. In Figure 4.3a, we can see that all CSPs are interconnected in a 1D array called a daisy chain. Figure 4.3b shows us that CSPs are connected by their four neighbours. At last, the tree and hybrid topologies introduce hierarchical levels (see 4.3d).

We will now briefly discuss the advantages and disadvantages of each available topology[6].

- **Daisy-chain:** This topology has the advantage that the hardware design can be reused, as the amount of input and output is the same for every node. On top of that, this topology is easy to scale, and adding a node to the existing ones is rather convenient. If a particular system needs fully distributed processing, this topology allows this without any problem. On the other side, the daisy chain topology can be an issue for applications with tight latency requirements. The number of hops from source to destination scales linearly (N) with the number of nodes.
- **2D-lattice** This topology requires each node to have at least four inputs/outputs, making the implementation and synchronisation harder. On top of that, there is a need for a larger number of connections between the different nodes. The positive impact of the higher number of connections is the scalability ($\sqrt{N}$) and the possible rerouting in case of any failure.
- **multi-level tree:** In terms of scalability, this topology scores better compared to the lattice and daisy-chain. With the scalability of $log\mathcal{N}$, this topology is preferred for applications where latency is a critical factor. However, the drawback is that there is a single node *fusion node* which can become a bottleneck if traffic and the number of computations rise.


(a) Daisy-chain topology


(b) Lattice topology


(c) Tree topology

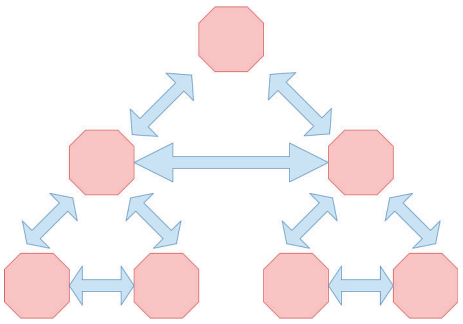

(d) Hybrid topology

Figure 4.3: Topologies in simulator

- **hybrid:** This topology differs from the multi-level tree in the connection between nodes on the same level. This reduces the number of hops and makes it a good candidate for many use cases as the processing is decentralised and latency is minimised.

To have an idea of how well the CSPs of the federation are interconnected to each other, the following procedure is followed. Notations are given in Table 4.2.

Symbol	Description
S	Set of all the CSPs in the federation
$t_{i,j}$	Topological distance from CSP i to CSP j
c	Topological center of the federation
a	Average distance from the center c to all CSPs
n	Number of CSPs in the federation
s_i	Sum of distances from CSP i to every CSP in a federation

Table 4.2: Notations latency

1. Calculate total sum of topological distances to every CSP, from every CSP

$$\forall i \in S, s_i = \sum_{j=1}^{n} t_{i,j}$$

2. Select the corresponding CSP that has the smallest sum as the center c

$$min\{s_1, s_2, ..., s_n\}$$

3. Determine the average distance a from this centre to the other CSPs

$$a = \frac{s_c}{n}$$

4. Normalise this average distance to a value between [0-1]
   (a) Determine minimum and maximum average distance by worst and best case scenario in particular configuration
   (b) Find a normalisation function that contains the points (0, maximum_distance) and (1, minimum_distance)

4.2.4 Data Rate and Packet Error Rate

Both data rate and packet error rate make use of the SINR metric. SINR is calculated as in equation 4.10, P stands for the power of the incoming signal of interest, I is the power of the interference, and N is the noise. Considering the previous calculations for WPT in Section 4.2.1, the SINR in our case results in Equation 4.11.

After obtaining the SINR, this value can again be normalised to a value between [0,1] by an arbitrary normalisation function that fits the current configuration. This mechanism will be demonstrated in Sections 5.2.1 and 5.2.2. As with real communication, there is a trade-off between data and packet error rates. This trade-off is mimicked by considering the values attributed to the utilities in the application's configuration files. The normalised SINR value is divided between the data rate utility and the packet error rate according to the defined values in the configuration files.

$$SINR_{dB} = 10 \times log_{10}\left(\frac{P}{I+N}\right) \tag{4.10}$$

$$SINR_{k(dB)} = 10 \times \log_{10}\left(\left|\frac{\sum^{L}\sqrt{P_{k,l}}w_{k,l}s_kh_{k,l}}{\sum^{L}(n_{k,l}+\sum_{k'\neq k}^{K}\sqrt{P_{k',l}}w_{k',l}s_{k'}h_{k,l})}\right|\right) \tag{4.11}$$

4.3 SimPy

SimPy is a process-based discrete-event simulation framework based on standard Python. The first version of this library was released on September 17th 2002, and has been updated many times since then. It has an efficient implementation and uses Python's generator capability. In SimPy, various scenarios can be simulated by built-in components like resources, processes, and events. SimPy has a well-developed API reference on its site accompanied by a view of topical guides and some interesting examples [33].

In the initial phase of this Degree-Project, a quick simulator was built to get some hands-on experience with discrete event simulation and understand the basics. The use of SimPy was not considered because of the possible slowness of the simulation and the consideration that later integration with other libraries would be needed. The uncertainty that these libraries might not integrate with the SimPy library led to the choice to leave SimPy as it was. The simulation part of the simulator was expected not to be very complicated as the main difficulty would become implementing the allocation structure.

In the later stages of the project, it became clear that collecting data for the simulation and keeping an overview of the different components in the simulation became more challenging than expected. The earlier choice to leave SimPy was then reevaluated, and it soon became apparent that the SimPy library's use simplified many aspects that first caused some difficulties. New discussions with the future users of the simulator led to the conclusion that the use of the library would not end up in integration problems. The use of SimPy resulted in code that was a lot more readable and organised.

This library enables the creation of events and, when triggered, puts them in the main event queue at a specified time t. The entities that can regulate the behaviour of the simulation are called processes. These processes are like functions that can yield certain events. When an event is yielded is stored as an object in memory and when triggered, it gets put into the event queue. This way of working is a lot more intuitive as each UE and federation in our setup can be seen as a separate entity with its own process. A certain process can, e.g., stop executing because of a yield statement that yields two events, one timeout event, which simulates the time it takes for the user to proceed to its next trajectory point, and one reconfiguration event, which will be triggered on reconfiguration. Whatever event that will be triggered first will continue the process where it last stopped, and different actions can be taken depending on which event triggered it. That is because when an event is yielded in a particular process, the processes *_resume()* will be added to its callbacks. If a non-timeout event gets triggered in another process, the event will be added to the event queue, and its callback will be executed when processed. It is possible to define as many processes as required, and they can all interact with each other through the shared event loop. The SimPy library has many more options, such as interrupts, resource containers, preemptive resources, and shared events. More information can be found on their well-designed website [33].

A wide variety of other packages and libraries have been used in the simulator to ease the development and avoid 'reinventing the wheel'. An overview is given in the Appendix A.1.

4.4 Process communication

Whenever UEs move, appear or disappear, a change of utility might be perceived by the rest of the UEs in the system. Sometimes these changes may lead to an increase in utility for a specific UE. Other times, it results in a decrease. Whenever the utility of a UE or a federation becomes critical, the current allocation needs to be adapted to counteract the drop in utility. What critical means is defined by the different thresholds and utility evaluation methods in the configuration file. An adaptation of the allocation can be made in two ways, either by a dynamic change of CSPs to a different federation or by making a whole new allocation and "resetting" the system. Only the second option is considered in this project, triggering a new configuration. This is mainly done to reduce complexity in this simulator, as deciding when to make a dynamic change or perform a complete reconfiguration is not straightforward. This decision opens up a bunch of new research questions which might be studied with the help of this simulator in the future. This reconfiguration mechanism is a first reason why different entities within our system should be able to communicate with each other and therefore need processes.

Additionally, to make the simulation as authentic as possible, it is trivial to mimic the communication of the different entities in the configuration in a certain way. Using the SimPy library eases how properties can be added to a specific communication link. When it comes to delays, the **env.timeout** function enables to yield a process for a specified amount of time. Combining this function with a particular probability distribution gives the user many possibilities to model a communication system in its preferred way. The most important functionality used in this simulator is the earlier mentioned SimPy process. A small simplified code sample is presented below to better understand how such a process works.

```python

def reallocation_process(self, env):

    while True:
        realloc_done = False
        check_event = env.timeout(10)
        reconf_events = [f.reconfig_event for f in self.allocation.
    federations]

        result = yield AllOf(env, reconf_events) | check_event

        if check_event in result:
            possible_replacement = self.reallocate()

            if possible_replacement is not None:
                self.reallocate(possible_replacement)
                realloc_done = True

        if check_event not in result or not realloc_done:
            new_alloc = self.alloc.allocate(self.ues, self.csps, self
    .env)
            self.reallocate(new_alloc)
```

Listing 4.1: reallocation_process

In Listing 4.4, we can see a simplified example process implemented in the reallocation entity. The code is responsible for performing reconfigurations whenever necessary. We can see that the global event loop is passed as an argument to the process. Whenever the method **env.run()** is called somewhere in the program, each process connected to that env will start its execution. Within the while loop, we can see the creation of the **check_event** which is a timeout event automatically triggered and inserted in the queue at time (current time + 10). In the **reconf_events** array, all the reconfiguration events from the current federations are stored. The yield statement stops the execution of the process until either the simulator has proceeded to time (current time + 10) or all of the events in the array are triggered. Whenever the process resumes, the events that have been triggered will be stored in the result dictionary that stores all the triggered events. Depending on the results, this dictionary can later be questioned to modify the system's behaviour. The AllOf() expression only resumes the process if all the events in the array are triggered. If the _resume() callback from check_event is executed first, and only some of the events in the array are triggered, the result dictionary will not contain any element from the array.

4.4.1 User Process

During the initialisation of the simulator, each UE object is created when the configuration files are parsed. Whenever the simulation reaches the time where a new UE has to appear online, the spawning process starts the process of the UE (See Section 4.4.4). This simPy process is dependent on three kinds of events that can be triggered. A different event handler function will be called depending on what event is triggered. The events are the **arrival_event**, **waiting_event**, and the **utility_check_event**. A utility_check_event is another timeout event that gets triggered when created, and scheduled at a configured time from the current time in the simulator. This event checks the UE's utility and warns its federation when it has lost utility; this will be elaborated in Section 4.4.2.

The arrival and waiting events take care of the movement of each user. In the configuration files, each user can have a trajectory predefined which tells the user where it has to go and how long it has to stay on that spot before leaving to its following location.

Thresholds on the User Level

Different kinds of utility thresholds can be declared in the configuration files. Every application can define a threshold on different levels. Minimal user utility thresholds (WPT, latency, packet error rate, positioning, data rate) can be defined to show when a user loses utility at a certain time. Another option is to define a threshold of the combinatorial utility the user has to meet to get any functionality called the **User Threshold Level**.

4.4.2 Federation Process

Federation objects are created and destroyed each time a reconfiguration occurs. The main events this process responds to are the **utility_check**, **federation_reconfiguration** and the **ue_lost_utility**. The **utility_check** event is generated on a pre-determined frequency and lets the federation check its utility to avoid letting the application lose its service requirements. On top of that, a user can also let its federation know that it has lost utility through the **ue_lost_utility** event, which gets triggered by a user checking its utility and noticing it has lost utility. When this situation occurs, the federation checks its overall federation requirements, and if it notices a reconfiguration is needed, it triggers its **federation_reconfiguration** event. The other active federations can detect when this event is triggered and responds by triggering their **federation_reconfiguration** event. This is important for the correct behaviour of the allocation process, which is further explained in the following Section (4.4.3).

Thresholds on the Federation Level

A federation has the possibility to define a percentile of users that needs to meet the overall user utility threshold. Suppose the percentage of UEs that do not meet this threshold is lower than the percentile defined in the configuration file. In that case, the total utility of the federation drops to zero. Another option is that all the **user level thresholds** are met, but the overall threshold of the federation is below the **federation threshold level**. This can happen when the **user level threshold** is higher than the overall **federation level threshold**. Keep in mind that the other way around is also possible. There is much flexibility in the configuration of the thresholds. This is desired as the use cases of the REINDEER project have diverse requirements.

4.4.3 Allocation Process

The allocation process monitors the reconfiguration events mentioned in the previous paragraph. When it notices that each active federation has triggered its reconfiguration event, it is allowed to initiate a reallocation. The allocation object does this reallocation, which uses the configured reallocation algorithm. This reallocation will result in several new federations and consequently the killing of the previous ones. Apart from this mechanism, there is also another **utility_check** event implemented, which is also triggered on a predefined frequency. The occurrence of these events provokes a utility check of each active federation. Based on the utility results, the allocation process can decide to perform a supplementary reallocation. The point of this reallocation is to optimise the system's overall utility. As the system's state has probably changed since the last reconfiguration, these optimising utility checks will almost always result in a reconfiguration.

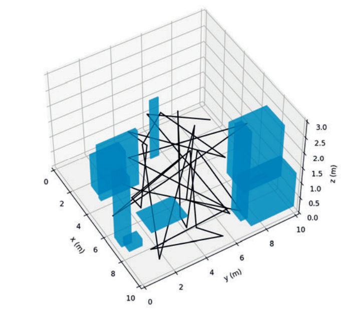
4.4.4 Spawning and Killing Process

These two processes are also implemented in the reallocation_entity, and they are solely responsible for handling the spawning and killing of UEs. The times when the UEs can come online and go offline need to be configured in the user's configuration file. This functionality is added to the simulator to mimic real-time behaviour in which specific devices can drop out for many different reasons e.g., a device out of battery, a user turning off the device, or the device being broken and unable to function anymore.

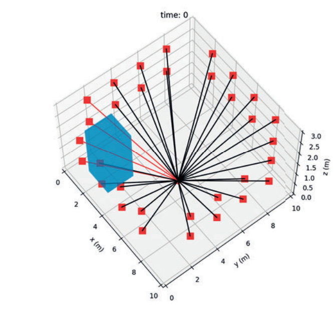
4.5 Integration of Objects and Ray Tracing

As mentioned in the introduction, objects are integrated into the simulator, adding a certain degree of complexity to the the code. The objects are defined in the **environment.yml** file in the configuration folder by defining an anchor point (x,y,z) and three dimensions (x,y,z), resulting in a rectangular volume. A simple version of ray-tracing is implemented as it should be avoided that UE have a trajectory that leads them through objects. The generated rays are checked if they do not intersect with any existing surfaces in the environment. If no intersections are found, the end-point of the ray can be seen as a valid waypoint in the trajectory. A valid trajectory is seen in Figure 4.4a. In Figure 4.4b a visualization is shown of the rays between a UE and the CSPs (red) on the wall. The rays that are coloured red indicate an intersection with the object.

This ray-tracing is also used in the calculations of the users signals developed to determine **SINR** and **signal power**. These two values than lead to the utility values of **WPT**, **data rate** and **packet error rate**. By checking if a ray between a UE and a CSP intersects with any object, shadowing can add an additional attenuation factor to the path loss. In terms of channel modelling, this means that if a ray does not intersect with an object, the UE has Line-of-sight (LOS) to the CSP the ray goes to. LOS means that a sending antenna can directly "see" the receiving antenna and vice versa.



(a) Objects with trajectory



(b) Ray Tracing with Objects

4.6 A virtual run

This section gives an overview of how the mechanism of running the simulator works. Many elements have been introduced, thus, going over the overall steps gives a better understanding of the total picture. To start the simulator from the command line the following command is required:

```
python -m src.simulator False False "/conf_directory"
```

The command contains three arguments. The first one indicates whether the simulator is run with debugging or not. If this argument is "true", the visualiser will display the configuration each time an event occurs. A "false" argument will simulate until the configured simulation time is over, all in one go. The second argument indicates whether the logger available in the project should be active or not. The newly created log file can then be inspected to understand better what happened in the simulation. The final argument must be the directory of the configuration files. The next steps are given below, together with a state diagram (Figure 4.5) for an overview.

1. **Initialisation:** The configuration files are parsed and converted into objects. When the User objects are created, their processes are not activated, as this only happens when they appear online. An allocator object is created and stored in the reallocation_entity object. This reallocation object also activates the **reallocation process** and the **spawning** and **killing process**.
2. **Allocating:** As soon as the spawning process spawns the first UE, the simulator goes to its allocation state. After the allocation, the simulator returns to the state where users move around.
3. **Users Moving:** When the simulator is running, this state is considered the "normal" state. UEs move around the simulated environment using the arrival and waiting for events (Section 4.4.1).
4. **Checking User Utility:** Whenever a user utility check event occurs, the simulator checks the utility of that user using its current location. If the user's utility is above the configured user threshold level, nothing happens, and the simulator goes back to state three (Section 4.4.1). If the user's utility has dropped, an additional utility check must be performed, and the simulator goes to state six.
5. **Checking Optimisation:** The transition to this state occurs whenever an optimisation event occurs. This is determined by the value given in the configuration files and is explained in Section 4.4.3. If a reconfiguration is decided, the simulator moves back to state 2.
6. **Checking Federation Utility:** The simulator can reach this state either by the process mentioned in 4 or by the occurrence of a federation utility check event explained in Section 4.4.2. If the federation finds out its utility has dropped, it will trigger a reallocation, and the simulator moves back to state 2.
7. **Starting or Stopping User Processes:** Whenever the spawning and killing processes spawn or kill a UE process, the simulator moves to state 2 passing through this state.

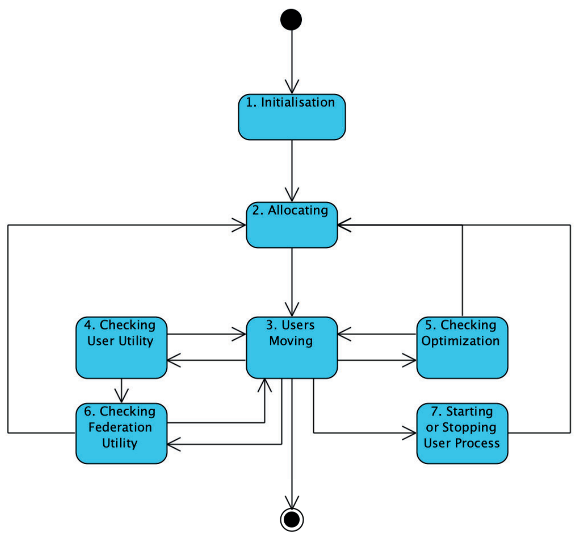


Figure 4.5: State diagram showing the different states of the simulator

4.7 Conclusion

In this chapter, the most important features and how they are implemented were highlighted;

- Channel model
- User mobility
- User arrival and departure
- Objects in the simulated environment
- Utilities
- Applications with utility preferences

These features form a good base for the simulator to perform some meaningful initial experiments. On the other hand, plenty of additional features can be added. These improvements to the simulator will be explained in Section 6.1.

Chapter 5

Testing

To verify the working of the simulator, it was tested in various ways. Combining the discrete event simulation and the utility calculations can produce results that are not very interpretative or easy to verify. That is because a simple allocation can already become hard to interpret in a configuration of even a few CSPs and UEs with different applications. The applications have a high degree of freedom in defining their utility preferences (WPT, latency, data rate, ...), leading to an allocation that might seem random with the naked eye. When we add the users' movements, the utility checks, and federation communication, this problem increases rapidly. That is why in this section, the proper functioning of the simulator is evaluated in two parts. In the first section, the event simulation will be tested, which implies examining the correct movements of the users, the communication between the federation hierarchy, and the reconfiguration mechanism. In the second section, the calculations of the utilities will be thoroughly tested.

5.1 The Discrete Event Simulation

The simulator needs to be tested in different scenarios to see if the system behaves correctly in time. As explained in the previous paragraph, this section will mainly focus on reviewing the correct scheduling and handling of the events. Therefore a different utility is introduced, which simplifies the calculations and helps keeping the allocation in the deployment manageable.

5.1.1 Simplified Utility

The simplified utility used in the following paragraphs is as follows; Suppose a federation exists that contains several CSPs and UEs. The utility of this federation can then be calculated by adding up the utility of each user. The utility for user k is calculated as follows: $u_i = \sum_j^L \frac{1}{d_{k,l}}$ with $d_{k,j}$ being the distance from UE k to CSP l. This measure for utility allows a more intuitive way of determining the importance of a CSP to a federation or UE, as the distance between two entities only determines it. It is essential to point out that the utility obtained by this definition does not return a normalised value between 0 and 1. This is in strong contrast with the utilities used in the actual allocator. The normalisation allows the highly combinatorial utilities to be manipulated as specified in the configuration.

5.1.2 Testing of User Process

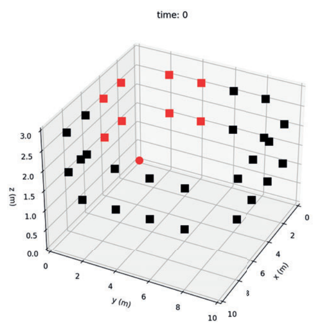


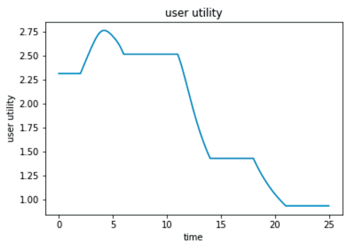
Figure 5.1: Demo configuration

y	arrival	wait	utility
0	0	2	2,31
4	6	5	2,52
7	14	4	1,43
10	21	2	0.93

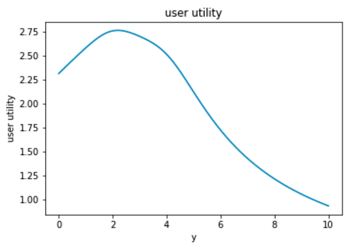
Table 5.1: User trajectory

In Table 5.1 we can see a possible trajectory from a user that is served by the red federation in Figure 5.1. The user moves from the figure's origin along the y-axis to the point (x=0, y=10, z=0). The utility this user experiences is demonstrated in time (Figure 5.2a) and space (Figure 5.2b).

Taking a closer look at the utility, it is clear that the users experience a rise in utility when it first moves along the y-axis, followed by a drop to a utility level of 0.93. The UE first moves relatively faster towards the four CSPs located on the (x=0)-surface than it moves away from the CSPs located on the (y=0)-surface. This explains the initial rise in utility experienced in Figure 5.2b. When the UE has reached y=2, the utility has reached its maximum. The UE then only keeps losing utility from the four panels on the (y=0)-surface and experiences no more gain in utility from the (x=0) panels. When y=4, the UE experiences a bigger drop in utility as it now moves away from all the panels in its federation. The same is shown in Figure 5.2a, but now the breaks of the UE result in a period of constant utility, as the utility is expressed with time as the independent variable.



(a) utility in time



(b) utility in space

Figure 5.2: User utility in time (a) and space (b)

5.1.3 Testing of Federation Process

In Figure 5.3, we can see the utilities of four users who make part of the same federation that they define and the same red CSPs of 5.1. Each users leaves (x=0, y=0, z=0) to (x=0, y=10, z=0) on different time. The utility rises before starting the descent, as shown in Figure 5.2a. When the user's utility drops below 2.13 (defined in the configuration file), the user loses utility and tells this to its federation. The percentile of users needing minimum user utility is 49%, also defined in the configuration file. We can see that when three out of the four users lose utility, the total utility of the federation drops below the required percentile (25% < 49%) and goes to zero. This demonstration shows that even if a federation still meets its average required federation utility (average of users utility), it could lose functionality because of the additional requirements. This way of handling utilities might be handy to configure in some use cases, e.g. if a group of sensors is sending the same data to a database. It is acceptable that a few UEs lose connection as long as a certain percentage of the devices are still available.

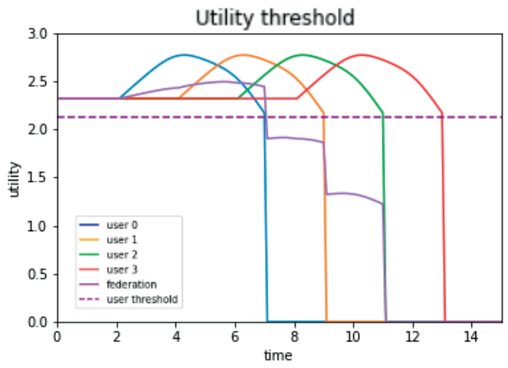
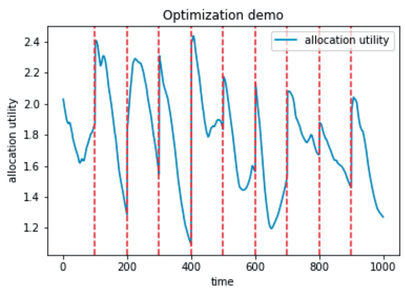
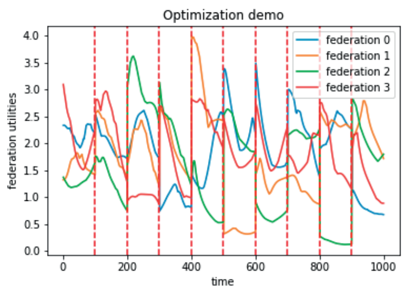


Figure 5.3: Threshold demonstration

5.1.4 Testing of Allocation Process



(a) Federation utilities



(b) Allocation utility

Figure 5.4: Results optimisation

A simple experiment with a configuration consisting of 32 CSPs, eight UEs and four applications(=four federations) demonstrates the optimisation mechanism. All the UEs have a random trajectory of 20 waypoints and a speed of 0.1 $\frac{m}{s}$. There are no configured thresholds, making the optimisation the only possible reason for reconfiguration. The simulation ran for 1000 STUs, and the allocation was checked every 100 STUs. The reallocations are triggered every time. This is visible in Figure 5.4b. This result confirms that every time the total utility was checked and compared, a new allocation had a higher utility than the previous one. This adds up because the movements of the UEs make them go to different locations. In terms of utility, these locations are less favourable than their starting location, wherefore the allocation was initially made. It is essential to know that the used algorithm for the reconfigurations does not guarantee the highest overall utility of all possible configurations. The reconfigurations are only decided when the reconfiguration (if made at that point in time) has a higher utility than the currently existing one.

5.1.5 Testing of communication mechanism

A new experiment is presented to test if the communication between a UE to the reallocation entity is behaving as expected. The configuration now consists of 4 UEs, and each UE has their separate application. This results in the creation of four federations each time a reconfiguration happens. The users randomly move through the configured environment. The UEs are configured to check their utility every STU. If the utility of the UE drops, it should notify the federation object. This federation object would then notify the other federations, as it would seem that the percentage of satisfied UEs is lower than the configured 100%. As soon as every federation has been notified, the reallocation entity will perform a reallocation. Figure 5.5 demonstrates this behaviour, UE zero, two and three all lose utility at a given time. At the next point in time where the utility of the UEs is measured, the utility has made a big jump. This significant jump indicates a reconfiguration. We can conclude that this part of the communication mechanism works as intended.

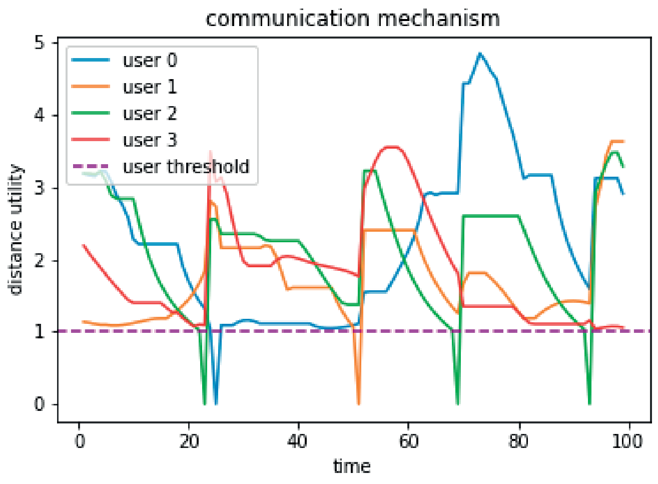


Figure 5.5

5.2 The Utilities

The second major part of this thesis project is the actual allocation of the different panels to the users and their application within a configuration. This allocation is done by allocation algorithms that coordinate the whole process based on the previously explained utilities. Several example configurations will be elaborated to demonstrate the working of the different utilities and the effect on some allocation scenarios.

5.2.1 WPT

To test the WPT calculations, we take a look at how the received signal power of a UE varies in space as the UE moves away from a wall of 50 CSPs (16 antennas per CSP), evenly spaced on the (y=0)-wall with heights of 1 and 3. The start position of the UE is (x=5,y=0,z=2). This UE will move along the y-axis to (x=5,y=10,z=1.5), and its signal strength will be measured along the way. The demo configuration is shown in Figure 5.6. The experiment is done four times, with a different CSP transmit power configured each time. The noise term from Equation 4.5 has been neglected.

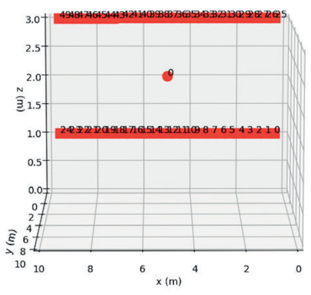


Figure 5.6: WPT configuration

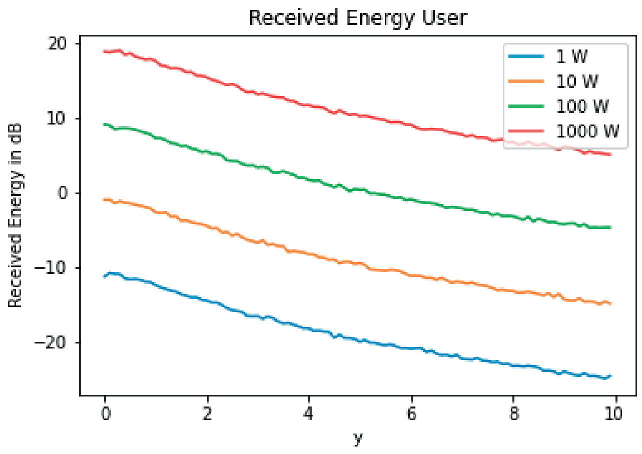


Figure 5.7: Results of WPT testing

In Figure 5.7, the received energy by the UE is shown. The received power decreases as the distance between the CSPs increases. The different coloured curves show us the influence of the transmit power per CSP, the reference power in Equation 4.7 is 1 Watt. We can see a difference of 10 dB between the curves. This corresponds to the expected $20log_{10}(\sqrt{10}) = 10dB$, derived from Equations 4.11 and 4.7. Calculations by hand have been conducted to see if the code explained in Section 4 gives the expected results.

5.2.2 Positioning

The positioning is demonstrated by yet another configuration shown in figure 5.8. There are 100 CSPs evenly placed on every wall at the height of 2.5; these CSPs will be added incrementally to the configuration to see the influence on the largest horizontal measured gap and the positioning utility, now only determined by the azimuth angle. The CSPs are sorted and added by id, which means that a new

CSP will be added right next to the previously added one. This results in the CSP square being created edge by edge. The gap will be measured for a UE placed on location (x=5,y=5,z=0).

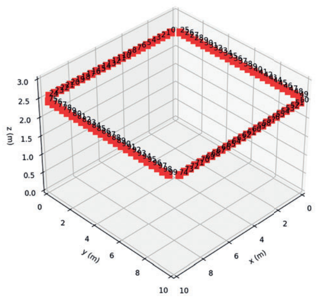
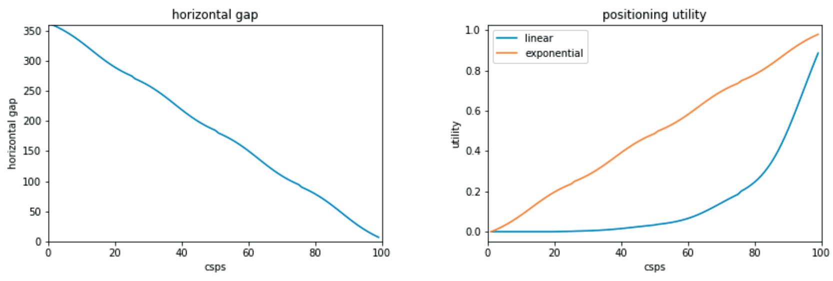


Figure 5.8: Positioning setup



(a) Biggest horizontal gap in federation

(b) Positioning utility

Figure 5.9: Results of positioning testing

The results show that the largest horizontal gap follows a linear downhill slope. It decreases by the amount of CSPs that are added to the federation. The line is not perfectly straight because adding a CSP in a corner does not have an equal impact on the gap as adding one in the middle of a wall (geometry). Figure 5.9b is added to show that the normalisation of the positioning utility, and actually any other utility can be adapted in a preferred way. The orange curve shows a linear normalisation as seen in Equation 5.1. That equation ensures that an angle gap of 360° results in a utility of 0, and an angle gap that approximates 0 will result in a utility of almost 1. The blue curve shows an exponential normalisation that follows Equation 5.2. The corresponding parameters are given in Table 5.2 and are obtained by performing curve fitting with an exponential function, some arbitrarily chosen (gap, utility) pairs, and the points (gap=1°, utility=1) and (gap=360°, utility=0).

$$horizontal\_utility = 1 - \frac{1}{360} \times hor\_angle\_gap \tag{5.1}$$

Parameter	Value
a	1.005331
b	0.01776996
c	0.01756831
d	-0.01756831

Table 5.2: Parameters logarithmic equation

$$horizontal\_utility = a - \frac{b}{c}(1 - e^{d*hor\_angle\_gap}) \tag{5.2}$$

5.2.3 Signal to Interference plus Noise Ratio (SINR)

Experiment 1a and 1b: Configuration

The next value of our simulator that will be tested is the SINR, a metric that is used to obtain the **data rate** and **packet error rate** utility of a UE or federation (see Section 4.2.4). Exact calculations to check if numeric results correspond to the ones given by the calculations in Section 4.2.4 have been conducted.

This experiment should show us the influence of the number of entities in a federation on the Signal to Interference (SIR) experienced by a UE placed in the middle of the environment (x=5,y=5,z=0). The SIR has the same characteristics as equations 4.10 and 4.11, except that the noise term is neglected. In both experiments, the CSPs are equipped with 16 antennas and have a transmit power of 100 Watt each.

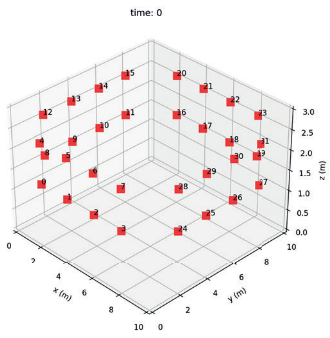
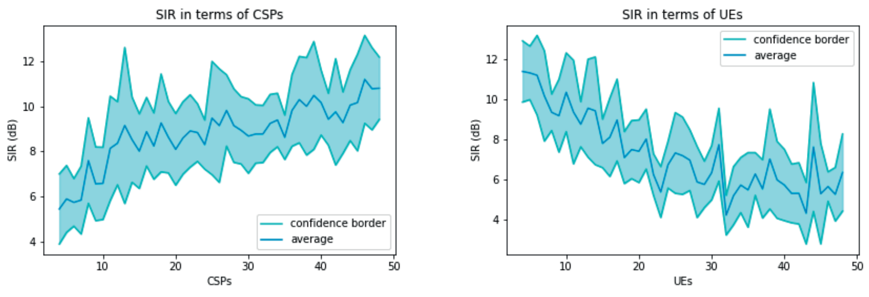


Figure 5.10: Performance configuration

In a experiment 1a, CSPs are added to a federation that contains eight UEs (including the monitored ue_0, locations in Appendix A.3.2). The addition of CSPs happens by id, which means that first, the lower row of a wall will be added, CSP by CSP. After that, the second row will be added to the configuration. This happens to every wall only when the previous wall is completed. The final CSP configuration has a total of 48 CSPs, evenly spaced on the walls (12 per wall, two rows per wall, six per row) on heights of 1.5 and 2.5.

A second experiment(1b) shows the influence of adding more UEs to a federation that already contains 32 CSPs. The configuration is the same as in Figure 5.1. The UEs are randomly placed on locations in the physical environment, increasing from four to 48. Exact locations of the UEs can be found in Appendix A.3.2.



(a) SIR in terms of CSPs **(b)** SIR in terms of UEs

Figure 5.11: Results of SIR experiments

Both tests have been run ten times, producing an average curve with confidence borders(95%) of the graph. The results are not identical for every test, as different channel vectors are generated every time. Resulting in different SIR values.

Experiment 1a: Results

In Figure 5.11a we can see that the SIR, in general, improves when more CSPs are added to the federation. This is a consequence of more CSPs transmitting coherently to the monitored UE, using the precoding described in Section 4.2.

Experiment 1b: Results

Figure 5.11b shows us that more UEs in a federation with the same amount of CSPs should in general give us a worse SIR. This can be explained by Equation 4.11, as the term in the denominator is now a sum with more terms. Another interesting thing to notice in Figure 5.11b is the increasing variance of the SIR. The interference caused by the new added UEs is highly dependent on the particular channel vector and other UEs locations. This is one reason why it is important to carefully select which CSPs and UEs are put into the same federation.

Both tests are conducted to demonstrate the general intuitions regarding SIR in terms of adding CSPs and UEs to a federation. The exact results are hard to interpret as the generation of random values highly influences the SIR results. The demonstration of the SIR behaviour will have the same trend as the SINR, as there is only a constant noise term introduced in the denominator of the SIR equation.

Experiment 2: Configuration

In the second experiment, 50 CSP are placed on a wall (y=0) (evenly spaced, in two rows (z=1;z=3, see Figure 5.6). All CSPs have a transmit power of 100 Watt and are equipped with 16 antennas. One UE (x=5,y=0,z=2) will move from this particular wall to the other side of the room (x=5,y=10,z=0), while the other seven randomly placed UEs will stay in the same position. To draw any valid conclusions from this experiment, the Rayleigh random values ($CN(0,1)$) are generated with a fixed seed. Because of this, the generated values will always be the same, and the movement only influences the SIR.

Experiment 2: Results

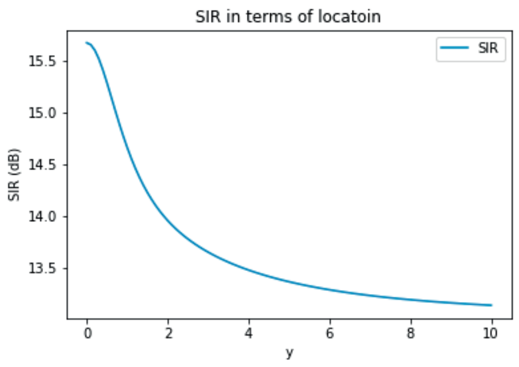


Figure 5.12: SIR in terms of location

As seen in Figure 5.12, the SIR is decreasing from approximately 15.5 dB to 13.5 dB. This is indeed the result we expected, as the SIR should go down in a logarithmic way as the UE moves away from its attributed panels. Because the distance to the CSPs increases a little slower in the beginning compared to the end (UE is moving on height z=2, see Figure 5.6), the curve shows deviance in the start of the curve. It is important to notice that changing the transmit power in the configuration files will not affect the SIR experienced by a UE. Both the signal and interference in Equation 4.11 would decrease with a lower reference power, resulting in the same SIR. If the noise term of Equation 4.5 is not neglected, a decrease in transmit power will result in a lower SINR. Both signal and interference will still decrease, but the constant noise term would now make sure the SINR has a lower value.

All the results from the previous experiments indicate that our channel model is correctly implemented. The simulator now has WPT, data rate and packet error rates that valuable characteristics that mimic real-world behaviour. The exact channel models specific for RW are yet to be concluded.

5.2.4 Wireless Power Transfer vs Positioning

A configuration designed to demonstrate the correct working of some utilities is shown in Figure 5.13. At position (x=5, y=5, z=0) two UEs are placed in the same position, that is why the red UE is not visible in Figure 5.13. Both UEs got assigned different applications, resulting in creating two different federations (red and blue). The application of the blue federation only values the **WPT** utility and does not need any other kind of utility. The same goes for the red application, but the only valued utility there is **positioning**. Additionally, 16 CSPs are placed on the walls of the room in a way that it is clear which panels should be assigned to what UE.

As **WPT** benefits from more panels to generate a strong signal, it would make sense that this federation gets assigned as many panels as possible. For the **positioning** utility, spatial diversity results in the highest value. The distributed panels would therefore be of higher value to that federation compared to the clustered ones. The result (Figure 5.13) shows us that the **shrinking algorithm** has done a good job assigning the right panels to the correct federation, considering both utility functions.

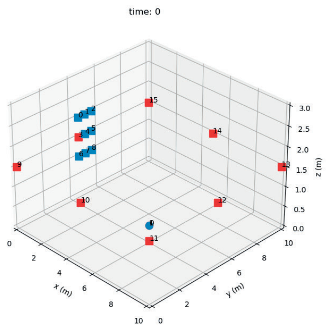
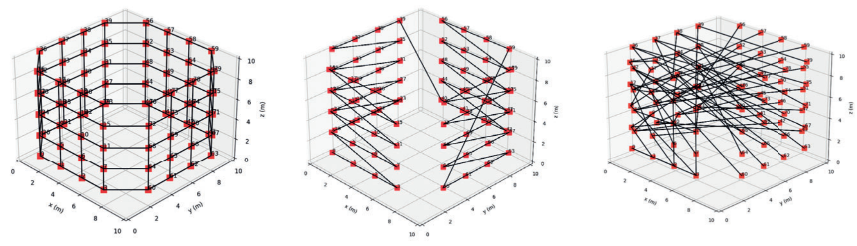


Figure 5.13: Resulting federations

5.2.5 Topologies and Latency Utility

Another demo configuration is presented to see if the latency utility is correctly implemented in the simulator. This demo contains a lot more CSPs compared to previous configurations. That is because it is more convenient to demonstrate the influence of the different configurations mentioned in 5.11. The experiment will examine the average distance from a CSP to any other CSP following the scaling presented in Section 4.2.3. The results will also show how the latency utility behaves when different topologies are used in a federation.

Configuration



(a) Lattice topology **(b)** Daisy-chain topology **(c)** Tree topology

Figure 5.14: Visualisation topologies

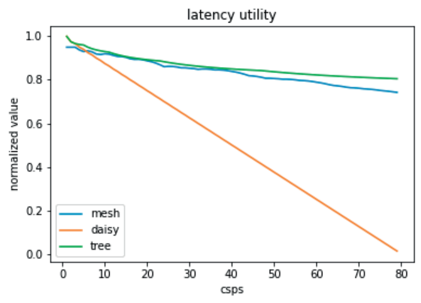
The configurations in Figure 5.14a are presented to test if the latency utility behaves as expected. In Figure 5.14a the **lattice** configuration is shown, as well the daisy-chain in Figure 5.14b and the tree topology in Figure 5.14c. Notice that the tree configuration looks very chaotic as the tree is built and sorted by id, resulting in connections of CSPs that are far apart from each other. CSPs are namely added by id, and only on the next wall when the previous one is completely filled. The CSP zero is the tree's root, and the highest id's form the leaves. This tree implementation is not realistic regarding real-life deployments of the RW structure. However, it can confirm the correctness of the code in terms of latency calculations. To use a realistic tree configuration, a topological distance matrix should be provided in the configuration files. No method is implemented to generate a realistic tree topology, taking the physical locations of the CSPs into account.

80 CSPs are evenly spaced on the four different walls of a cubical environment with dimensions 10x10x10. As the implementation of the latency utility is implemented as being independent of the position of its UEs, the utility is only determined by which CSPs are in the federation and how good their topological connection is. The test consists of incrementing the amount of CSPs attributed to a virtual federation by ID. Subsequently the **average distance**, **latency utility** and **utility input** (see Section 4.2) of the federation is calculated. The normalisation of the utility input to an actual latency utility value should always be adapted to the configuration used. The formula used to obtain this value in this configuration is given below (Equation 5.3).

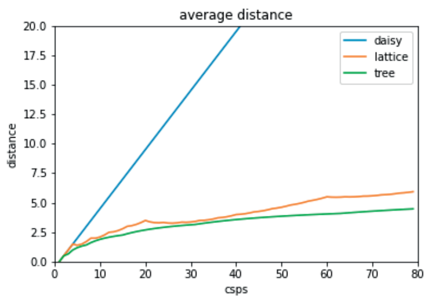
$$utility\_value = 1 - 0.05 \times utility\_input \qquad (5.3)$$

Results

The results of the experiment are shown below in Figure 5.15. When the corresponding topology is used, the average distance follows the expected scaling tendency. On top of that, we can see that the chosen utility input (Section 4.2.3), combined with the conversion Equation 5.3 produces utility values that cover the whole [0-1] range.



(a) Latency utility



(b) Average distance

Figure 5.15: Latency results

5.3 Performance

To evaluate the simulator's performance, we will look at the impact of the number of UEs, CSPs, and utility-check frequency on the execution time. The configuration is specified for each criterion under examination, and the results are subsequently discussed. As this section aims to evaluate the simulator's performance, the shrinking algorithm is replaced with a random allocation of the CSPs. This has been done because experiments have clarified that the shrinking algorithms' execution time was too dominant with increasing entities. The prominent factor that will determine the execution time is expected to be the amount UEs, as more UEs will lead to equally more UE processes and consequently more UE utility checks. These checks can also become more expensive as the amount of CSPs in the federation increases (utility calculations).

The program was run on an Apple MacBook Air, equipped with an Apple M1 chip. The device had eight cores and 8 GB of memory, and no other applications were running concurrently. The used version of Python was 3.10. A more detailed overview of the hardware and software specifications of the machine on which experiments were conducted is given in Appendix A.4.

Overall Configuration

This section evaluates the simulator's performance by recording the program's execution time. Firstly, the overall configuration and the relevant parameters used in this simulation run will be discussed. The start configuration serves as the foundation for the following experiments. The start configuration will be altered depending on what element is being investigated. These changes are mentioned below.

- **CSPs:** There are 32 CSPs evenly placed inside the environment(x=10, y=10, z=3). In figure 5.10 we can see their positions, together with the corresponding ID's of the CSPs.

- **UEs:** The users have a speed of 0.1 and a random trajectory of 20 waypoints. The following procedure determines each user's application; UE.id % 4 (number of applications). The users will be online from time=0 till the end of the simulation to decrease the influence of random generated spawn and killing times (triggering a reconfiguration) on the execution time.

- **Applications:** There are four applications in the configuration, which means a default of four federations will exist if all applications are used. All the thresholds that can be configured are reduced to a level of 0.00. That is to avoid a reconfiguration being triggered by one of the users or federations, as the goal of this test is to evaluate the simulator's performance and not that of the random allocation. The triggering of the reallocation events can be evaluated in future tests.

- **Objects:** The standard configuration does not contain any objects, as the ray-tracing would be too much of a burden on the simulator.

- **Parameters:** Other relevant parameters are mentioned below. The reallocation_entity utility period is made very large to avoid reconfiguration by optimization.

  – Simulation time : 1000

  – UE utility check period: 60

  – federation utility check period: 180

  – ray_tracing: False

  – reallocation utility check period : infinite

5.3.1 Increasing UEs

Configuration

The UEs will be incrementally added to the configuration on a random location within the following intervals (x=[0-10], y=[0-10],z=[0-1.5]). The test is run ten times with an increasing number of UEs from 4 to 100, and the average execution time in seconds together with a confidence interval of 95% is plotted.

Results

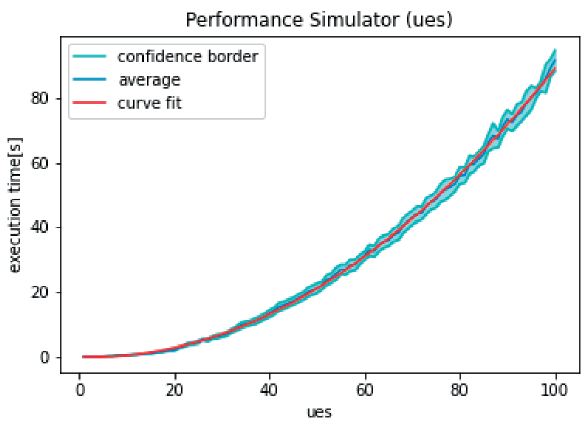


Figure 5.16: Performance results with Increasing UEs

The results are shown in figure 5.16. It can be concluded that the execution time of the simulation is not linear and has a quadratic gradient as specified in equation 5.4 with parameters given in table 5.3. This result can be explained by the fact that apart from more movements and utility checks, the execution time of each utility check also increases.

$$a * num\_ues^2 + b * num\_ues + c = execution\_time \tag{5.4}$$

Parameter	Value
a	0.0077088916432401 79
b	-0.1999845691162747
c	2.016100302243215

Table 5.3: Parameters quadratic equation

5.3.2 Increasing CSPs

Configuration

The same configuration is applied in this experiment as in the previous section, except for the CSPs and the UEs. The CSPs are now incrementally added to the walls in an evenly spaced way, with the number going from 4 to 104. The number of UEs is now eight and will stay constant throughout the test. The configuration of each UE stays unchanged, just as the approach of objects, ray tracing, and other parameters. The test is run ten times with an increasing number of CSPs from 4 to 104. The resulting graph shows the average execution time, accompanied by a confidence interval of 95%.

Results

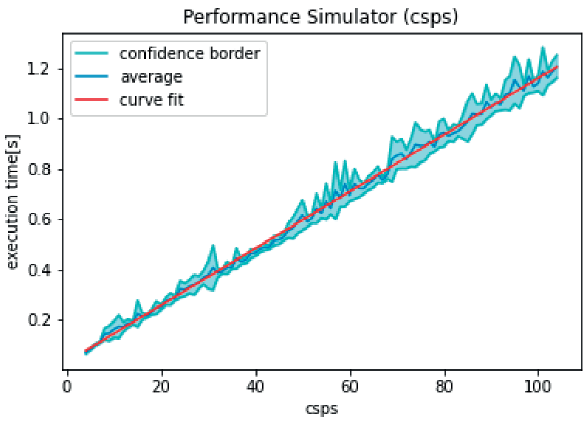


Figure 5.17: Performance results with Increasing CSPs

The results are shown in Figure 5.17. Here, the execution time (seconds) of the simulator is linear with the amount of CSPs in the federation. This result confirms that performing a utility check is linear to the configuration's amount of CSPs. The linear slope is described by Equation 5.5 with parameters given in table 5.4

Parameter	Value
a	0.0112912...
b	0.0311562...

Table 5.4: Parameters linear equation

$$a * num\_csps + b = execution\_time \tag{5.5}$$

5.3.3 Increasing Utility Check Interval

Configuration

The CSP are configured in the same way as mentioned in Section 5.3, the applications and the UEs are equivalent to those in Section 5.3.2. There are still no objects in the environment and the paramaters remain unchanged from section 5.3.

Results

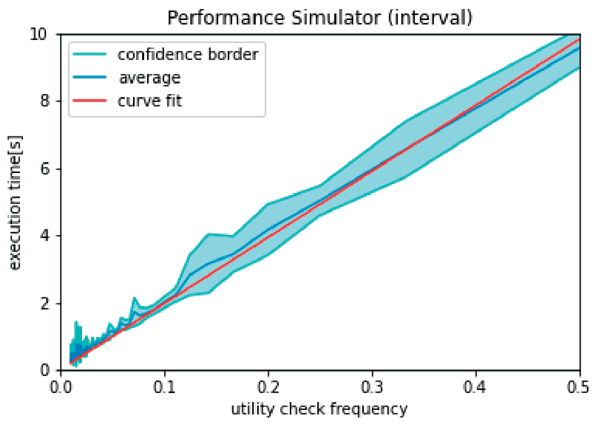


Figure 5.18: Performance results with increasing interval

The last results show that the higher the UE utility check frequency, the higher the execution time (seconds) for a certain simulation. This behaviour is expected as a higher frequency leads to more utility checks; these are costly for the simulator and determine the execution time. The approximated curve and equation is given in figure 5.18 and equation 5.6, together with the corresponding parameters in table 5.5. The confidence interval is created with a confidence of 95%.

Parameter	Value
a	19.62027162

Table 5.5: Parameters logarithmic equation

$$execution\_time = a \times x \tag{5.6}$$

5.4 Conclusion

In this chapter, the different parts that make up the simulator were tested. Section 5.1 demonstrated with simplified utilities that the scheduled events and SimPy processes function as expected. On top of that, Section 5.2 provided results that show the correct implementation of the used metrics and utilities. Although the performance was not a vital factor of the simulator, Section 5.3 gave us some interesting insights; Not all entities and parameters have the same influence on the performance. The number of configured UEs has a bigger impact than the number of CSPs on the simulation's execution time. Configuring a larger utility check interval results in fewer utility evaluations, which also influences the simulation performance in a non-negligible way. In all three tests, there is low variance. This is good as the simulator's performance is not very dependent on the placement of the entities(UEs and CSPs), the movements and resting times of the UEs, or other random generated values.

Looking at the overall execution time of the simulator for 100 UEs, it seems to average out at around 90 seconds or 1.5 minutes. It is not yet entirely clear what STU will be used in the simulator, as this highly depends on the distance metric used in a specific configuration. Nevertheless, this execution time seems reasonable to get some results for federation allocation.

Chapter 6

Future Work and Conclusion

As this thesis presents a first version of the simulator and the implementation time frame consisted of one semester, many improvements and additions can be made in several ways. This Section will give an overview of the most important ones together with a final conclusion on the Master's project.

6.1 Future Work

6.1.1 Predicting Utility Drops

The current implementation of the simulator does not allow entities within the system to predict when their utility drops. Utilities are evaluated, and reconfiguration signals are only sent when the utility values are below a certain threshold. However, this is often not the desired operation, especially not with critical applications on the devices. UEs and federations should be able to predict when their utility will drop, allowing the system to anticipate and perform a reconfiguration upfront.

6.1.2 REINDEER Use Cases

As this thesis is a part of the REINDEER project, it is relevant to keep in mind the Use Cases presented in Section 2.4. After the extension with more low-level communication data and ray tracing, it becomes interesting for the researchers inside the REINDEER project to simulate the use cases they presented and see the impact of different allocation algorithms in a more realistic scenario. To do so, a few extra things will need to be added to the simulator.

Different Areas

Many of the Use Cases need a more complex physical environment; *Augmented reality for sport events*, *Patient monitoring with in-body and wearable sensors* and *Wander detection and patient finding* are a few examples of such. Currently, only one simple room can be configured where users can move around. The physical environment should be able to be configured such that different, preferably inter-connected, areas can be simulated. These areas should also have the possibility to possess different properties, e.g., open-air vs roofed, flat surface vs sloping ground, or certain rooms with restricted access for specific UEs.

Realistic Movement Patterns

Closely related to Section 6.1.2 are the different movement patterns of the UEs that will need to be added. These movement patterns are very Use-Case specific and could significantly impact how the simulation behaves. The current configuration generator only allows random movement patterns to be generated, with the option to add rest-time when a UE arrives at a particular location. To obtain the movement patterns, the configuration generator could be extended with a generator method for each use case that generates patterns that reflect the movements of the concerned UEs.

Integration of Technical Requirements

It has already become clear that throughout this thesis, the approach to allocating CSPs to federations is by using utilities. This term utility expresses how valued a CSP is to a UE or federation. To connect the simulator more to the real world, it is desired to introduce more low-level concepts. These concepts would allow a more detailed understanding of the actual data communication between the entities in the system. This is another way of integrating the proposed Use Cases in the simulator, as more technical requirements could be configured such as **Traffic volume density**, **carrier frequency**, **end to end latency** and much more.

6.1.3 Visualisation

The current visualization implemented in the simulator is mainly designed to ease debugging and get a general overview of what is happening in the simulator. As the 3D modelling capabilities of the matplotlib library are relatively limited, future versions of the simulator should be able to represent the real world more realistically and lift the aesthetic level of the demos. This could be achieved by utilizing game engines that have become very accessible.

6.1.4 Optimization

In terms of optimization, many improvements can be made. As the simulation time of the simulator was not a critical requirement for this thesis, the focus was mainly on the correct functioning of the code. Improvements in the utility calculations and simulation processes can be made to speed up the execution time and optimize performance. Also, data structures in the current code version were mainly chosen because of convenience, as speed was not considered a critical factor. These elements might be adapted in future releases if performance becomes a bottleneck.

6.2 Conclusion

In this thesis, a high-level simulator was designed to simulate and evaluate federation orchestration algorithms. To do so, the RW technology that will be used in the REINDEER project was explained, after which the simulator's design was elaborated. Additionally, the simulator was tested in different scenarios to verify correct behaviour.

It is possible to simulate static as well as dynamic situations. Users can come online and disappear again. Additionally, they can move around the configured physical environment and occasionally take a rest at different waypoints. On top of that, objects can be added to the simulator to represent the simulated environment better. To obtain more realistic allocations based on real-world behaviour, utilities were introduced. These functions are designed to mimic the preferences of the applications according to their technical requirements and express the preference of certain federations to the CSPs. They do this while avoiding heavy calculations that would be introduced using ray-tracing and other more low-level characteristics.

References

[1] Wikipedia, “Spherical coordinate system — Wikipedia, the free encyclopedia,” http://en.wikipedia.org/w/index.php?title=Spherical\%20coordinate\%20system&oldid=1084232641, 2022, "[Online; accessed 25-April-2022]".

[2] REINDEER. Resilient interactive applications through hyper diversity in energy efficient radioweaves technology. WebPage. REINDEER consortium. [Online]. Available: https://reindeer-project.eu

[3] G. Callebaut, J. Van Mulders, G. Ottoy, D. Delabie, B. Cox, N. Stevens, and L. Van der Perre, “Techtile – open 6g r&d testbed for communication, positioning, sensing, wpt and federated learning,” 2022. [Online]. Available: https://arxiv.org/abs/2202.04524

[4] B. Sihlbom, M. I. Poulakis, and D. Renzo, “Reconfigurable intelligent surfaces: Performance assessment through a system-level simulator,” p. 7, November 2021. [Online]. Available: https://arxiv.org/abs/2111.10791

[5] E. Basar and I. Yildirim, “Simris channel simulator for reconfigurable intelligent surface-empowered communication systems,” Koç University. IEEE, November 2020. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9282349

[6] O. Edfors, R. Brazil, H. Petautschnig, G. Callebaut, T. Feys, L. V. der Perre, E. G. Larsson, O. Edfors, E. Fitzgerald, L. Liu, J. R. Sanchez, W. Tärneberg, P. Frenger, B. Deutschmann, T. Wilding, and K. Witrisal, “Initial assessment of architectures and hardware resources for a RadioWeaves infrastructure,” Jan. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5938909

[7] G. Callebaut, W. Tärneberg, L. Van der Perre, and E. Fitzgerald, “Dynamic federations for 6g cell-free networking: Concepts and terminology,” *arXiv preprint arXiv:2204.02102*, 2022, to appear.

[8] L. V. D. Perre, E. G. Larsson, F. Tufvesson, L. D. Strycker, E. Bjornson, and O. Edfors, “Radioweaves for efficient connectivity: Analysis and impact of constraints in actual deployments,” vol. 2019-November, 2019.

[9] "Home of rf and wireless vendors and resources," webpage, Wireless World, 2012. [Online]. Available: https://www.rfwireless-world.com/Terminology/Difference-between-cellular-network-and-cell-free-network.html

[10] EU H2020 REINDEER project. (2021) REsilient INteractive applications through hyper Diversity in Energy Efficient RadioWeaves technology (REINDEER) project - Deliverable 1.1: Use case-driven specifications and technical requirements and initial channel model. Visited on 2021-07-26. [Online]. Available: https://reindeer-project.eu/D1.1

[11] S. Gunnarsson, J. Flordelis, L. Van der Perre, and F. Tufvesson, "Channel hardening in massive MIMO-a measurement based analysis," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2018, pp. 1–5.

[12] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5g wireless networks: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016.

[13] J. Zhao, "A survey of intelligent reflecting surfaces (irss): Towards 6g wireless communication networks," 2019. [Online]. Available: https://arxiv.org/abs/1907.04789

[14] R. Alghamdi, R. Alhadrami, D. Alhothali, H. Almorad, A. Faisal, S. Helal, R. Shalabi, R. Asfour, N. Hammad, A. Shams *et al.*, "Intelligent surfaces for 6g wireless networks: A survey of optimization and performance analysis techniques," *IEEE access*, 2020.

[15] S. Gong, X. Lu, D. T. Hoang, D. Niyato, L. Shu, D. I. Kim, and Y.-C. Liang, "Toward smart wireless communications via intelligent reflecting surfaces: A contemporary survey," *IEEE Communications Surveys and Tutorials*, vol. 22, no. 4, pp. 2283–2314, 2020.

[16] M. Di Renzo, M. Debbah, D.-T. Phan-Huy, A. Zappone, M.-S. Alouini, C. Yuen, V. Sciancalepore, G. C. Alexandropoulos, J. Hoydis, H. Gacanin *et al.*, "Smart radio environments empowered by reconfigurable ai meta-surfaces: An idea whose time has come," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–20, 2019.

[17] E. Basar, "Transmission through large intelligent surfaces: A new frontier in wireless communications," in *EuCNC 2019*, Koç University. IEEE, 2019, pp. 112–117. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8801961

[18] S. Hu, F. Rusek, and O. Edfors, "The potential of using large antenna arrays on intelligent surfaces," in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8108330

[19] E. Fitzgerald, W. Tärneberg, F. Tufvesson, O. Edfors, L. V. der Perre, D. Delabie, R. Sarvendranath, E. G. Larsson, K. Witrisal, B. J. B. Deutschmann,

A. Reial, U. Muehlmann, M. Truskaller, and J. F. E. Rivas, "Interactive applications in need of 6g: Technical requirements facilitated by sub-10 ghz radioweaves," 2021.

[20] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. PEARSON, 2014. [Online]. Available: https://www.scribd.com/document/529375972/Jerry-Banks-Et-Al-Discrete-Event-System-Simulation-Pearson-2014

[21] S. Bouveret, Y. Chevaleyre, and N. Maudet, "Fair allocation of indivisible goods," 2016. [Online]. Available: https://web.archive.org/web/20150906224807id_/http://www-poleia.lip6.fr/~maudetn/teaching/12-comsoc-main.pdf

[22] H. Aziz, B. Li, H. Moulin, and X. Wu, "Algorithmic fair allocation of indivisible items: A survey and new questions," 2022. [Online]. Available: https://arxiv.org/abs/2202.08713

[23] E. Gerding, A. Perez-Diaz, H. Aziz, S. Gaspers, A. Marcu, N. Mattei, and T. Walsh, "Fair online allocation of perishable goods and its application to electric vehicle charging," May 2019. [Online]. Available: https://eprints.soton.ac.uk/432204/

[24] T. Kalinowski, N. Narodytska, and T. Walsh, "A social welfare optimal sequential allocation procedure," vol. 13. Universität Rostock and NICTA and UNSW, April 2013. [Online]. Available: https://arxiv.org/abs/1304.5892

[25] S. R. Sinclair, G. Jain, S. Banerjee, and C. L. Yu, "Sequential fair allocation of limited resources under stochastic demands," 2020. [Online]. Available: https://arxiv.org/abs/2011.14382

[26] M. Aleksandrov and T. Walsh, "Online fair division: A survey," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, pp. 13 557–13 562, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/7081

[27] A. D. Procaccia and J. Wang, "Fair enough: Guaranteeing approximate maximin shares." New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2600057.2602835

[28] M. Ghodsi, M. Hajiaghayi, M. Seddighin, S. Seddighin, and H. Yami, "Fair allocation of indivisible goods: Improvements and generalizations." New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219166.3219238

[29] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi, "On approximately fair allocations of indivisible goods." New York, NY, USA: Association for Computing Machinery, 2004. [Online]. Available: https://doi.org/10.1145/988772.988792

[30] A. Asadpour and A. Saberi, "An approximation algorithm for max-min fair allocation of indivisible goods," *SIAM Journal on Computing*, vol. 39, no. 7, pp. 2970–2989, 2010. [Online]. Available: https://doi.org/10.1137/080723491

[31] R. Project, "future release : Reindeer deliverable 3.2," https://reindeer-project.eu/results-downloads/, 2022.

[32] T. Marzetta, E. Larsson, H. Yang, and H. Ngo, *Fundamentals of Massive MIMO*, 1st ed. Cambridge University Press, 2016. [Online]. Available: https://books.google.se/books?id=Be08DQAAQBAJ

[33] S. Team. (2022, march) Overview - simpy 4.0.2.dev1+g2973dbe documentation. webpage. [Online]. Available: https://simpy.readthedocs.io/en/latest/index.html/

Appendix A

Extra material

A.1 Used packages

Library	Description
NumPy	NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
math	This module provides access to the mathematical functions defined by the C standard.
random	This module implements pseudo-random number generators for various distributions.
sys	This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
time	This module provides various time-related functions.
itertools	This module implements several iterator building blocks inspired by constructs from APL, Haskell, and SML.
glob	The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order.
unittest	The unittest unit testing framework was originally inspired by JUnit and had a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.
SciPy	SciPy is a free and open-source Python library used for scientific computing and technical computing.
Matplotlib	Matplotlib is a cross-platform data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use Matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

Table A.1: Libraries with their description

A.2 Configuration examples

A.2.1 application_0.yml

```
id: 0
WPT:
    value: 0.04
    threshold: 0.00
POSITIONING:
    value: 0.11
    threshold: 0.00
LATENCY:
    value: 0.67
    threshold: 0.00
PACKET_ERROR_RATE:
    value: 0.13
    threshold: 0.00
DATA_RATE:
    value: 0.05
    threshold: 0.00
utility_evaluation: MEAN
threshold_user_level: 0.00
threshold_federation_level: 0.00
percentile_satisfied: 0.7364712141640124
max_num_users: 100
binary_evaluation: false
federation_check_interval
```

Listing A.1: application_0.yml

A.2.2 csp_0.yml

```
id: 0
coords: [2.0, 0, 1.4]
```

Listing A.2: csp_0.yml

A.2.3 ue_0.yml

```
id: 0
coords: [6.394267984578837, 0.25010755222666936, 0.4125439775536789]
application: application_0
speed: 0.1
online: 0
offline: 9999999999
utility_check_interval: 10
trajectory:
    - [1.395379285251439, 1.024951761715075, 1.1110016170015138, 8]
    - [0.8693883262941615, 4.2192181968527045, 0.044695829157105516,
    3]
    - [2.326608933907396, 6.020187290499804, 0.8418675944079195, 10]
    - [7.01324973590236, 4.195198209616588, 0.6738135694257804, 4]
    - [8.094304566778266, 0.06498759678061017, 1.2087288777492118, 6]
    - [3.4025051651799187, 1.5547949981178155, 1.4358196083101717, 5]
    - [1.022102765198487, 3.7992730063733737, 0.5384690707269426, 5]
```

Listing A.3: ue_0.yml

A.2.4 algorithm_configuration.yml

```
#general configuration
    ------------------------------------------------------------------
simulation_time: 1000
algorithm: "random"
csp_topology: "mesh"
ray_tracing: "false"
num_of_antennas_in_csp: 16
wavelength_constant: 20.0
random_seed: 42
transmit_power: 50
optimisation_interval: 10000

#simulated annealing parameter selection
    --------------------------------------------------
SIMULATED_ANNEALING:
  start_temperature: 100
  iterations: 100
  time_to_run: 10
  alpha: 0.90
```

Listing A.4: algorithm_configuration.yml

A.2.5 environment.yml

```
x: 10
y: 10
z: 3
objects:
    - [4, 4, 0, 3, 3, 3]
    - [3, 3, 0, 3, 3,3]
```

Listing A.5: environment.yml

A.3 Experiment 1a and 1b

A.3.1 Location of UEs in 1a

```
ID: 0 POSITION: (x= 5, y= 5, z= 0)
ID: 1 POSITION: (x= 2.4663, y= 5.6137, z= 0.3941)
ID: 2 POSITION: (x= 0.7099, y= 6.311, z= 0.3434)
ID: 3 POSITION: (x= 6.5737, y= 5.6523, z= 0.4746)
ID: 4 POSITION: (x= 1.7936, y= 9.2449, z= 1.1736)
ID: 5 POSITION: (x= 6.1926, y= 0.9339, z= 1.428)
ID: 6 POSITION: (x= 2.7294, y= 4.8564, z= 0.5833)
ID: 7 POSITION: (x= 5.2175, y= 6.5046, z= 0.9239)
```

Listing A.6: environment.yml

A.3.2 Location of UEs in 1b

```
ID: 0 POSITION: (x= 5, y= 5, z= 0)
ID: 1 POSITION: (x= 2.4663, y= 5.6137, z= 0.3941)
ID: 2 POSITION: (x= 0.7099, y= 6.311, z= 0.3434)
```

```
ID: 3 POSITION: (x= 6.5737, y= 5.6523, z= 0.4746)
ID: 4 POSITION: (x= 1.7936, y= 9.2449, z= 1.1736)
ID: 5 POSITION: (x= 6.1926, y= 0.9339, z= 1.428)
ID: 6 POSITION: (x= 2.7294, y= 4.8564, z= 0.5833)
ID: 7 POSITION: (x= 5.2175, y= 6.5046, z= 0.9239)
ID: 8 POSITION: (x= 1.1409, y= 0.6534, z= 0.7376)
ID: 9 POSITION: (x= 1.8149, y= 5.8533, z= 0.9522)
ID: 10 POSITION: (x= 8.5601, y= 4.1078, z= 0.4773)
ID: 11 POSITION: (x= 1.0665, y= 3.8136, z= 0.5384)
ID: 12 POSITION: (x= 0.3721, y= 9.9593, z= 0.1742)
ID: 13 POSITION: (x= 1.0796, y= 7.124, z= 0.6653)
ID: 14 POSITION: (x= 7.4097, y= 9.7181, z= 0.1376)
ID: 15 POSITION: (x= 2.3627, y= 1.4644, z= 0.2959)
ID: 16 POSITION: (x= 3.0958, y= 3.7668, z= 1.1875)
ID: 17 POSITION: (x= 3.8617, y= 0.8161, z= 0.3368)
ID: 18 POSITION: (x= 6.469, y= 2.8611, z= 1.2932)
ID: 19 POSITION: (x= 6.6784, y= 7.0954, z= 0.825)
ID: 20 POSITION: (x= 8.0735, y= 4.0609, z= 0.6803)
ID: 21 POSITION: (x= 5.1933, y= 1.4894, z= 1.3397)
ID: 22 POSITION: (x= 2.8763, y= 7.1414, z= 0.5195)
ID: 23 POSITION: (x= 2.4805, y= 2.6079, z= 0.3533)
ID: 24 POSITION: (x= 8.9133, y= 4.4854, z= 1.3757)
ID: 25 POSITION: (x= 2.55, y= 3.3026, z= 0.1186)
ID: 26 POSITION: (x= 7.0717, y= 0.0914, z= 0.7586)
ID: 27 POSITION: (x= 6.0517, y= 6.1387, z= 0.8934)
ID: 28 POSITION: (x= 3.9824, y= 3.1914, z= 1.1023)
ID: 29 POSITION: (x= 6.9676, y= 3.1891, z= 0.4504)
ID: 30 POSITION: (x= 7.1014, y= 4.4385, z= 0.2509)
ID: 31 POSITION: (x= 4.547, y= 4.8804, z= 0.8729)
ID: 32 POSITION: (x= 6.8657, y= 8.5996, z= 0.13)
ID: 33 POSITION: (x= 1.3721, y= 0.1135, z= 0.7446)
ID: 34 POSITION: (x= 7.5893, y= 3.7627, z= 1.1071)
ID: 35 POSITION: (x= 9.1743, y= 0.7499, z= 1.4957)
ID: 36 POSITION: (x= 0.9026, y= 2.3262, z= 0.3282)
ID: 37 POSITION: (x= 8.2881, y= 7.8873, z= 0.8671)
ID: 38 POSITION: (x= 5.7271, y= 2.3564, z= 0.4483)
ID: 39 POSITION: (x= 9.1343, y= 4.0717, z= 0.7267)
ID: 40 POSITION: (x= 6.5612, y= 9.5219, z= 0.9772)
ID: 41 POSITION: (x= 4.3371, y= 4.2941, z= 0.6435)
ID: 42 POSITION: (x= 1.6037, y= 5.4814, z= 1.3655)
ID: 43 POSITION: (x= 7.2665, y= 4.8193, z= 0.9956)
ID: 44 POSITION: (x= 5.8932, y= 5.042, z= 1.4458)
ID: 45 POSITION: (x= 6.9294, y= 0.4698, z= 0.5724)
ID: 46 POSITION: (x= 4.4556, y= 0.4385, z= 1.3474)
ID: 47 POSITION: (x= 8.3475, y= 1.9433, z= 0.272)
```

Listing A.7: environment.yml

A.4 Specifications

There were no other applications running on the system when the tests were conducted.

A.4.1 Hardware

- Model Name: MacBook Air

- Model Identifier: MacBookAir10,1
- Chip: Apple M1
- Total Number of Cores: 8 (4 performance and 4 efficiency)
- Memory: 8 GB
- System Firmware Version: 7429.81.3
- OS Loader Version: 7429.81.3

A.4.2 Software

- System Version: macOS 12.2.1 (21D62)
- Kernel Version: Darwin 21.3.0
- Boot Volume: Macintosh HD
- Boot Mode: Normal

A.4.3 Python

- Interpreter: Python 3.10

Printed by Tryckeriet i E-huset, Lund 2022

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2022-882
http://www.eit.lth.se