Modelling, Simulating and Emulating Distributed Applications in Swarms of Cyber-Physical Systems Deployed in Dynamic Networks

Bruno Chianca Ferreira

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *28/03/2023* **par :**

Bruno CHIANCA FERREIRA

Modelling, Simulating and Emulating Distributed Applications in Swarms of Cyber-Physical Systems Deployed in Dynamic Networks

---

JURY

Guthemberg SILVESTRE	Enseignant-Chercheur	Directeur de Thèse
Guillaume DUFOUR	Ingénieur de recherche	Directeur de Thèse
Valeria LOSCRI	Chargée de recherche	Rapporteuse
Abderrezak RACHEDI	Professeur des universités	Rapporteur
Fabrice THEOLEYRE	Directeur de recherche	Examinateur
Ernesto EXPOSITO	Professeur d'Université	Examinateur
André-Luc BEYLOT	Professeur d'Université	Examinateur
David SANCHEZ	Maître de conférences	Examinateur

École doctorale et spécialité :

EDSYS : Systèmes embarqués 4200046

Unité de Recherche :

Ecole Nationale de l'Aviation Civile (ReSCo)

Directeur(s) de Thèse :

Guthemberg SILVESTRE et *Guillaume DUFOUR*

Rapporteurs :

Valeria LOSCRI et *Abderrezak RACHEDI*

Résumé — La recherche sur les systèmes distribués a connu récemment un intérêt croissant pour les applications pour un environnent mobile, c'est-à-dire les algorithmes distribués déployés dans des réseaux dynamiques et sans fil. Cependant, la conception de ses systèmes présentent de nouveaux défis en raison de leurs topologies variables dans le temps, qui peuvent compromettre la performance et la sûreté des algorithmes distribués. L'objectif principal de ce travail est d'établir différentes manières d'étudier de tels systèmes via l'émulation et la simulation, et de proposer quelques stratégies pour déployer des applications en environnent mobile. Nous avons évalué différents paradigmes de flux de données, tels que le stockage distribué multi-nœuds, l'ordonnancement de tâches de calcul sur plusieurs nœuds, le contrôle décentralisé des essaims de drones et l'équilibrage de la charge de données.

Nous avons adopté une approche quantitative comme méthodologie de base, en utilisant la simulation et l'émulation pour collecter les données à analyser. Bien qu'il existe de nombreux simulateurs discrets dans l'état de l'art, aucun n'était adapté à l'émulation des systèmes répartis sur des réseaux mobile ad hoc. Certaines caractéristiques importantes manquaient, telles qu'un environnement de déploiement émulé à grande échelle où les prototypes coexistent avec les applications réelles et davantage d'options de contrôle de la mobilité. Ce travail propose MACE, un environnent qui permet l'émulation d'applications mobiles distribuées dans un environnement virtuel de sorte que les scénarios et les topologies composés par les nœuds mobiles sans fil peuvent être facilement modifiés. Comme l'émulation exige que les tests s'exécutent comme les vrais systèmes, le besoin d'un simulateur rapide s'est fait sentir. Nous avons donc conçu, mis en œuvre et validé un modèle fluide qui simule le flux de trafic dans les réseaux mobiles ad hoc. Avec ce modèle, nous avons finalement mis en œuvre un outil de simulation qui permet une analyse rapide des algorithmes et des paramètres et qui peut potentiellement être intégré dans des nœuds contraints pour une réévaluation de la mission en temps d'exécution. Le modèle fluide peut s'adapter à de grandes topologies avec des centaines de nœuds et peut réaliser des expériences en exécutant des charges de travail sous contrainte avec un temps de simulation plus court que l'horizon temporel de l'émulation. Il peut également être configuré avec des files d'attente de réseau limitées et non limitées, utiliser des modèles de mobilité et des lois de contrôle mixtes, et modéliser différentes applications fonctionnant dans la couche d'application avec injection de trafic synthétique. En étudiant différentes lois de contrôle, nous pouvons réduire la probabilité de partitions durables du réseau et améliorer l'équilibre du trafic pour réduire la formation de goulots d'étranglement qui entravent les performances de l'application.

Notre travail englobe également la proposition de certaines applications liées au domaine de la gestion du trafic des systèmes d'aéronefs sans pilote (de l'anglais Unmanned Aircraft System Traffic Management, UTM), telles que des algorithmes optimisés pour la bordure du réseau des essaims de drones qui peuvent être potentiellement utilisés pour les systèmes distribués de détection et d'anti-collision UTM. En outre, nous proposons également une architecture de suivi de position distribuée pour l'espace aérien à très basse altitude en utilisant la réplication, où nous avons pu mesurer de faibles latences de bout en bout même avec un nombre élevé de répliques.

Mots clés : systèmes distribués, systèmes cyber-physiques, modèle fluide, simulation, émulation, systèmes critiques, utm

Abstract — The research field in distributed systems has witnessed a recent growing interest in wireless mobile distributed computing, i.e. distributed algorithms deployed in dynamic networks. Such systems introduce new challenges due to their time-varying topologies, which hinder the performance and safety guarantees of algorithms that are fundamental building blocks of more complex distributed algorithms. The main focus of this work is to establish different ways to study such systems via emulation and simulation and to propose some techniques to deploy applications in dynamic networks. We evaluated different data flow paradigms, such as many-to-many distributed storage, one-many computational offloading, and many-to-many decentralized swarm control and data load-balancing.

We adopted the quantitative approach as a cornerstone methodology, using simulation and emulation to collect data for analysis. Even though there are many discrete simulators in the state-of-the-art, there was a gap for emulation tools applied to mobile ad hoc computing. Some important features were missing, such as a full-scale emulated deployment environment where prototypes coexist with off-the-shelf applications and more options for mobility control. This work proposes MACE, a framework that enables the emulation of mobile distributed applications in a virtual environment so that the scenarios and topologies composed by mobile wireless nodes can be easily modified. Since emulation requires that the tests run in wall time, the need for a fast simulator arose. Therefore, we designed, implemented and validated a fluid model that simulates traffic flow in mobile ad hoc networks. With this model, we finally implemented a simulation tool that enables fast algorithm and parameter analysis and can potentially be embedded in constrained nodes for in-flight mission re-evaluation. The fluid model can scale to large topologies with hundreds of nodes and could complete experiments running stress workloads with simulation time shorter than the simulation time horizon. It can also be configured with bounded and unbounded network queues, use mixed mobility models and control laws, and model different applications running in the application layer mixed with synthetic traffic injection. By studying different control laws, we can reduce the probability of enduring network partitions and enhance the traffic balance to reduce the formation of bottlenecks that hinder the application's performance.

Our work also encompassed the proposal of some applications related to the domain of UTM, such as optimized edge-assisted offloading algorithms for swarms of UAVs that can be potentially used for UTM Distributed Detect and Avoid systems. Moreover, we also propose a distributed position tracking data layer for very low-level airspace using State Machine Replication and could achieve low end-to-end latencies even with a high number of replicas.

Keywords: distributed systems, cyber-physical systems, fluid model, simulation, emulation, critical systems, utm

Acronyms

ADS-B Automatic Dependent Surveillance–Broadcast.

AI Artificial Intelligence.

AODV Ad Hoc On-Demand Distance Vector.

API Application Programming Interface.

APRS Automatic Packet Reporting System.

AR Augmented Reality.

ATM Air traffic management.

BATMAN Better Approach to Mobile Adhoc Networking.

BVLOS Beyond Visual Line of Sight.

CAP Consistency Availability Partition.

CPS Cyber-Physical System.

CPU Central Processing Unit.

DDAA Distributed Detect And Avoid.

EMANE Extendable Mobile Ad hoc Network Emulator.

EN Edge Nodes.

FANET Flying Ad-Hoc Network.

FIFO First In First Out.

FLOP Floating Point Operations.

FLOPS Floating Point Operations Per Second.

GNSS Global Navigation Satellite System.

GUI Graphical User Interface.

HPC High Performance Computing.

IoT Internet of the Things.

LTE Long Term Evolution.

MAC Mobile Ad-Hoc Computing.

MAC Medium Access Control.

MACE Mobile Ad hoc Computing Emulator.

MANET Mobile Ad-Hoc Network.

MCC Mobile Cloud Computing.

MEC Mobile Edge Computing.

ML Machine Learning.

OLSR Optimized Link State Routing Protocol.

OS Operating System.

PER Packet Error Rate.

PID Proportional Integral Derivative.

QoS Quality of Service.

RAM Random Access Memory.

RF Radio Frequency.

ROM Read-Only Memory.

RT Round Trip.

SAA Sense And Avoid.

SDN Software Defined Network.

SINR Signal to Interference and Noise Ratio.

SMR State Machine Replication.

SoB System on a Board.

SoC System on a Chip.

SoM System on a Module.

SPP Self-propelled Particle.

sUAV Small Uncrewed Aerial Vehicle.

TCL Technical Capability Levels.

TCP Transmission Control Protocol.

TDMA Time Division Multiple Access.

UAS Uncrewed Aircraft System.

UAV Uncrewed Aerial Vehicle.

UDP User Datagram Protocol.

UE User Equipment.

USS UAS Service Suplier.

UTM UAS Traffic Management.

UUV Unmanned Underwater Vehicle.

VANET Vehicular Ad-Hoc Network.

VLOS Visual Line of Sight.

VM Virtual Machine.

VR Virtual Reality.

WSN Wireless Sensor Networks.

Contents

Author's rights and use conditions by third parts

Chapter 1

Introduction

Contents

The research in distributed systems has witnessed a recent growth in interest in distributed algorithms deployed in dynamic networks, i.e. mobile distributed applications on devices connected via wireless networks. Such systems introduce new challenges due to their time-varying topologies, which often hinder the liveness and termination of algorithms, making it challenging to provide reliability and safety, which are critical desired guarantees. Those are also essential when running algorithms, such as population counting [78], membership control [121], and topology radius measurement [53], which are fundamental components of more complex distributed applications. Many of these algorithms assume having a view of the network that allows all nodes to know the topology's global state. For instance, when performing reliable broadcasting, it is common for the broadcaster to expect confirmation from the receivers so that it is known which nodes successfully and reliably delivered the message. In [62] and [61], for example, there was an effort by the authors to establish an algorithm that enables the nodes to build a view of the network that consists of the nodes currently partaking in the broadcast. To do so, some assumptions must be considered, which are challenging to comply within dynamic networks.

Dynamic topologies have nodes deployed in such a way that the connections between them are unstable. That could be caused due to the nature of the network connections or the physical environment. For example, because the nodes themselves are constantly changing their relative position to each other or because the physical medium where the messages are propagated is *noisy*. Some examples of the latter are wireless links enduring

heavy interference, nodes geographically far from each other connected via a long series of relaying nodes, and satellite links with nodes fading *Beyond Visual Line of Sight (BVLOS)*. All these factors related to dynamic network behaviour introduce challenges and research opportunities that is worth exploring.

1.1 Context and Motivation

Different paradigms and applications can exploit mobile computing architectures. Although the concept is not new [50], the challenges have evolved with the adoption of new communication technologies and advancements in computing capacity [142, 143]. Furthermore, as applications become more data-centric (e.g., edge and *Machine Learning (ML)* applications that require processing large amounts of sensor data), the amount of data needed to be transferred increases. So, the challenges' focus shift to how to make data available to applications whilst maintaining the ability to comply with the performance goals [176, 165]. The most common paradigms adopted for latency-sensitive and privacy-aware applications are the *Edge* and *Fog computing*, where data is preprocessed at edge servers before being sent to the cloud. In addition to enabling lower latencies by preprocessing data, edge servers can also be leveraged to increase data safety. For instance, a *Cyber-Physical System (CPS)* swarm connected via ad hoc networks can be sent on a mission to survey and collect data, which must be stored on a distributed store during the mission before offloading to an edge cloudlet [55]. In other applications, collected data must be distributed among all nodes for sensor data fusion [30]. Such swarms can assume different topologies depending on how the mission was planned and evolved during runtime, creating challenges and opportunities to tame the network flow and thus reduce congestion and delay.

Concerning the communication infrastructure, the main characteristics of mobile swarms are how the mobility and network density affect their connectivity [180, 73]. Therefore, how to implement and simulate mobility using different paradigms such as simple mobility models, and realistic motion control models, was an important topic in this thesis. Since the network functions of such nodes during their runtime are usually complementary, data flows can be arriving, traversing and leaving the node simultaneously. Due to the limited communication range, as shown in Figure 1.1 and comprehensively surveyed in [166], communication and routing often happen in a multi-hop manner. As a consequence of these limitations, some nodes with a strategic location in the topology might experience excessive traversing traffic flow.

Therefore, it is valuable to study the behaviour of the network flows for different topologies and applications, since this can be used to prevent unwanted congestion by employing different deployment strategies or control laws that act directly on nodes' mobility to balance the traffic flow across the topology. The limited communication range associated with the nodes' mobility also increases the probability of creating network partitions. Consequently, it also affects the availability and consistency of distributed applications deployed in such topologies as described by the CAP theorem [57]. Such mobile nodes connected via ad hoc wireless interfaces are commonly referred to as *Mobile Ad-Hoc Networks (MANETs)*, or more specifically *Flying Ad-Hoc Networks (FANETs)* when in the domain of flying nodes, and *Vehicular Ad-Hoc Networks (VANETs)* in the vehicular domain. The introduction of the term *swarms*

is more common when there is also an interest in introducing aspects of swarm intelligence and control laws, as a way to provide some level of autonomy to system. The challenges and opportunities related to MANETs and swarms will be further discussed in Chapter 2.

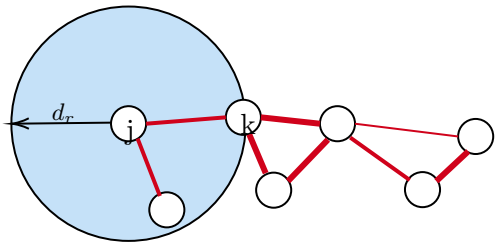


Figure 1.1: Topology Model Representation Based in Distance Between Nodes (a.k.a Unit Disk Radio).

1.1.1 Other Hardware Limitations

Besides the communication, other hardware limitations also need to be taken into consideration. Since devices used in swarms of CPS are deployed remotely and are mobile, they must have an energy storage unit. They are commonly comprised of a battery, mainly Lithium Polymer batteries in the most modern devices, a protection circuit and a battery charger. As of today, those energy storages are limited; therefore, many applications and protocols have energy consumption as one of the leading design goals.

Another essential component is the computing unit, where the main firmware (*Operating System (OS)* and other low-level software) is run. The firmware controls the main logic and any other software that controls other possible embedded hardware. The type of unit depends on the application, but usually, it is a micro-controller [1], a *System on a Chip (SoC)* [2] or a *System on a Module (SoM)* [3]. Since these nodes have limited dimensions and energy storage, it is crucial to consider computing units that supply just enough computing power needed for the application and use the least amount of energy possible. Another constrained component in mobile CPS is the memory unit. It is divided into volatile, used in runtime by the firmware, or persistent, where the nodes can store new states. As the compute unit, due to footprint limitations, memory also needs to be limited to the strict necessary to avoid extra energy consumption.

Considering these hardware limitations and the challenge to embedded complex algorithms required by CPSs, some applications chose to use computational offloading to edge cloudlets or other devices. This is particularly interesting in applications that require low latency, increasing the feasibility of applications that are unable to be deployed in constrained devices. The current and future ubiquity of edge servers, owned to the strong presence of telecommunication equipment used by 4G and 5G services, allow the deployment of such applications with benefits to the companies and to the users.

[1] Small factor computers often with 8 or 16-bit microprocessor and some other modules

[2] Devices that often concentrate all the required modules to build a computer inside one *die*. So *Central Processing Unit (CPU)*, *Random Access Memory (RAM)*, *Read-Only Memory (ROM)*, and other communication peripherals are on a single chip

[3] Devices with a small footprint that can fit on a module and might be connected to a bigger carrier board that make more IO ports available to the user.

1.1.2 Definitions

Although the definition of distributed computing diverges among different authors [159, 128], in this thesis, we will refer to distributed systems or computing systems according to the definition of [128]:

Definition 1.1
Distributed Computing *- "As we have seen, the essence of distributed computing is different. It is on the coordination in the presence of "adversaries" (globally called environment) such as asynchrony, failures, locality, mobility, heterogeneity, limited bandwidth, restricted energy, etc. From the local point of view of each computing entity, these adversaries create uncertainty generating non-determinism, which (when possible) has to be solved by an appropriate algorithm."*

Definition 1.2
Parallel Computing *- "...parallel computing addresses concepts, methods, and strategies which allow us to benefit from parallelism (simultaneous execution of distinct threads or processes) when one has to implement f (x). The essence of parallel computing lies in the decomposition of the computation of f (x) in independent computation units and exploit their independence to execute as many of them as possible in parallel (simultaneously) so that the resulting execution is time-efficient. Hence, the aim of parallelism is to produce efficient computations. This is a non-trivial activity which (among other issues) involves specialized programming languages, specific compilation-time program analysis, and appropriate run-time scheduling techniques."*

Additionally, we will refer to computation acceleration via parallel processors/cores or cluster/grid computing as parallel computing, which in contradiction is also called *Distributed Computing* by [159].

1.2 Swarms and their Applications

From a high-level point of view, this thesis studies aspects related to computer networks and distributed systems. These subjects are extensive, and following the previously given hints, the reader will notice that the focus is given to more specific areas of these fields. From the computer networks' point of view, attention is given to wireless networks, specifically ad hoc networks. Swarms, as used in this document, borrows the term from nature and refers to computer topologies composed of nodes which are similar in physical construction and software implementations. They run algorithms that, together, as a whole, aim to solve a higher-level application problem. When considering a swarm, nodes can be deployed in a way where they can only react to the topology changes, i.e. their position in space over time changes only due to an external force, or they can act proactively, when considering *Self-propelled Particles (SPPs)* for instance [19]. The latter can change their position in space by creating a thrust vector in the desired direction. SPPs, due to their ability to move through space by their own will, allow the development of cooperation between the nodes to enhance their capability to fulfil their mission. This cooperation has been well observed in nature

[19] for instance, in swarms of insects, bees, ant colonies, whale pods, and many of such cooperation, often referred to as swarm or bio-inspired intelligence, have been copied and recreated in swarms of CPSs [106, 84]. We aim, among other things, to study the impact the flow of information has on distributed applications running the aforementioned wireless ad hoc networks. The scenario where these applications will run is on swarms of cyber-physical systems. As this research work evolved, there was a broad spectrum of applications that could be deployed in such a paradigm. In this section, we introduce some of these applications that can benefit from high-performance distributed applications running on dynamic networks.

1.2.1 Earth Observation, Remote Sensing and Surveillance

Earth observation is a field constantly pushing state-of-the-art technology boundaries since it benefits directly from every opportunity that can put human eyes where it was previously impossible. Starting a long time ago with hot air balloons, evolving to large aircraft and large satellites as seen in Figure 1.2, and more recently small factor satellites [108, 36] and *Uncrewed Aircraft Systems (UASs)* [169, 174].

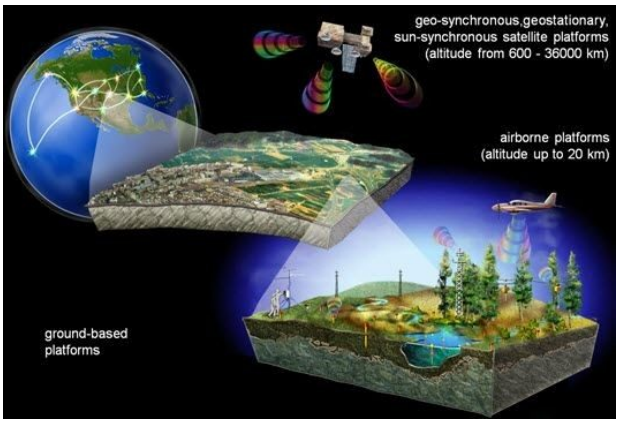


Figure 1.2: Aircraft and Satellite Earth Observation. [4]

Hence, the possibility of having distributed data processing is valuable, and the system only needs to offload processed data when passing by the ground station. The swarms can also drastically increase the coverage area [171], decreasing the necessary orbits to reach the same coverage. Another application for swarms of UASs is surveillance [160, 141], as different optimization algorithms can be used, other goals such as improved area coverage, reduced energy consumption, or execute faster intruder detection. As many other technologies, it has been used also for military operations as early as 1849, when a swarm of hot air balloons was used by the Austrian military on an attempt to bomb the besieged Venice. An artistic representation of the event is show in Figure 1.3.

[4] https://www.npoc.ch/en/about/earth-observations.html

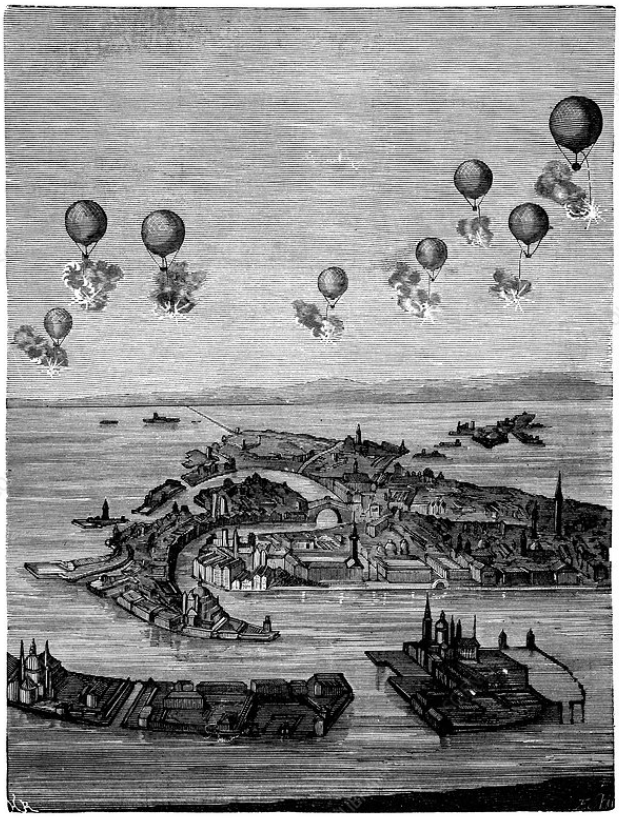


Figure 1.3: Pencil Drawing Depicting the Austrian Balloon Swarm used for Military Action in 1849, by French civil engineer Max de Nansouty (1854-1913). [5]

1.2.2 Agriculture, Industry and Mining

Algorithms similar to those used for surveillance can also be used in agriculture [5, 8, 167]. Computer vision can be leveraged to infer the soil quality and state, be used to spot unwanted invasive species in the field or even be used for precision mapping. As a result, those applications can be used to improve precision farming, as shown in Figure 1.4a and eventually helping manual labour with automatic identification, early warnings and unbiased recommendations and pesticides application, as shown in Figure 1.4b. Such swarms are also used in massive industrial and mining compounds for not only surveillance, but also to evaluate production with various types of sensors. It can also be used in mining facilities so detect in advance possible fatal accidents such as landslides and rupture of dams. The crucial aspect of the swarms in such scenarios is the ability of carrying different types of payload, and the capability of autonomously navigate remote and dangerous areas without human intervention.

[5] https://www.sciencephoto.com

(a) UAV Swarm used in Precision Agriculture Mapping. [6]

(b) UAV Swarm used for Pesticides Application. [7]

Figure 1.4: UAV Swarms used in Precision Agriculture

1.2.3 Underwater Exploration and Sensing

Following the same principle, *Unmanned Underwater Vehicle (UUV)* swarms and sensor networks [32] have been used to support different applications. Earth-wise, the oceanic coverage is larger than the ground coverage, and in some parts much deeper than the highest mountain ranges. Because of that, the ocean is commonly referred as being as unexplored by us as space. Therefore, such swarms can be leveraged to deep exploration, where the human presence is dangerous, and they also have been used for academic purposes to explore marine life and for topographic mapping purposes. They are also used for asset monitoring by the Oil & Gas industry, coastal monitoring by defence forces and rescue operations in hard to access locations. Figure 1.5 shows an example of underwater communication network used by UUVs.

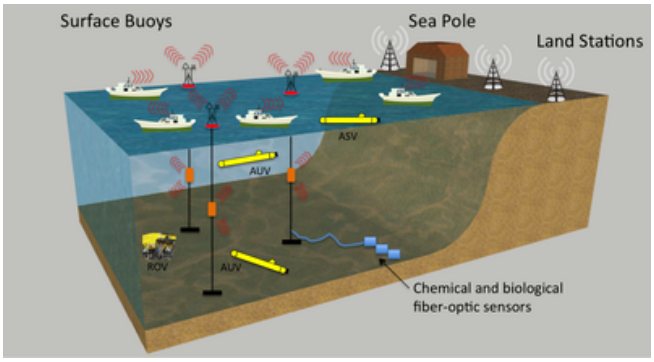


Figure 1.5: Underwater Communication Network. [8]

Underwater communication has some challenges in the MAC/PHY layers when compared to its surface counterparts. Usually lower frequencies in the audible spectrum are used,

[7] https://vertical-master.ch/drone-en-agriculture/
[7] https://groupechd.fr/blog/agriculture-des-tracteurs-bientot-remplaces-par-des-drones/
[8] http://www.tec4sea.com/

which can disturb marine life and only allows lower throughput and high propagation time. Furthermore, it is affected by seasons, depth, water temperature and salinity, making it hard to predict latencies. For this reason, a complete new set of *Medium Access Control (MAC)* protocols have been developed [15].

1.2.4 Planetary Exploration and Astronomy

Planetary exploration has witnessed many missions using various types of robots such as probes and rovers. It is natural to imagine that the robustness added by CPS swarms can be leveraged in such application [155]. Not only they would be able to cover larger areas with mission crafted formations, but failures of individual nodes would not jeopardize the mission due to the added fault tolerance. Such swarms can be used not only on the surface but also as nanosatellites orbiting an asteroid. The authors in [156] propose an asteroid exploration concept where a main spacecraft could be deployed together with a nanosatellite swarm, whose measurements are combined using Kalman Filters. Besides exploration, the robustness and flexibility of swarms in dynamic environments has also applications in astronomy [95, 35, 25]. In these cases, the main rational behind is to scale the radio astronomy observation by combining the signals captured by each node so that they work as a big telescope. This concept has been used on earth since the 80's with the Very Large Array telescope, shown in Figure 1.6, and can now be leveraged in space, avoiding the atmospheric turbulence.

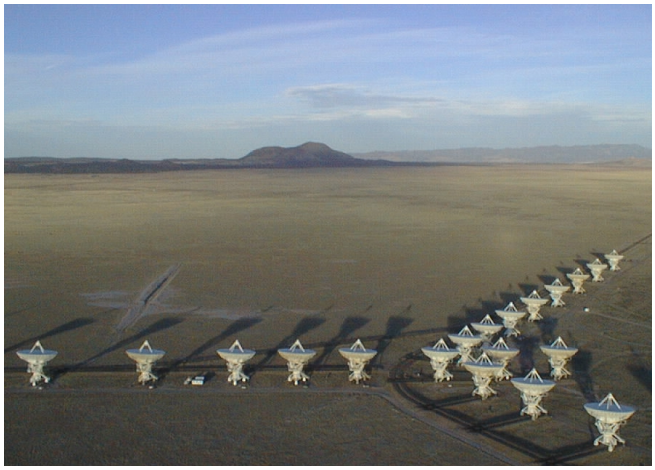


Figure 1.6: Very Large Array Project in New Mexico USA. [9]

Another application field for such swarms are related to the *UAS Traffic Management (UTM)* systems, from which some aspects were approached in this thesis in Chapters 5 and 6. All the aforementioned applications are emerging and the solutions, specially in distributed coordination and data exchange, have not yet been consolidated. Therefore, communication in distributed applications running in dynamic topologies was one of the main motivation for this thesis.

[9] source: http://www.aoc.nrao.edu/epo/puente/views/vlaviews.index.html

1.3 Contributions

The main focus of this work was to study data exchange in distributed applications deployed in dynamic networks. That included a major goal to establish methodologies to study such systems via emulation and simulation and propose some techniques concerning deploying applications in dynamic networks.

1.3.1 Fluid Model Simulator (dNetFlow)

Even though there are tools in the state-of-the-art that can be used to study distributed systems and computer networks, they often scale poorly and are not adapted to study mobile constrained systems. As an alternative, some advances were made towards simulating discrete systems with continuous quantities by using fluid models. Therefore, to solve the challenge of studying network flows in distributed applications deployed in dynamic networks, we propose a fluid model which is presented, tested and validated in Chapter 3 and published as a conference paper [45]. It simulates traffic flow in mobile ad hoc networks and allows us to implement a simulation tool that enables fast algorithm and parameter analysis and can potentially be embedded in constrained nodes for in-flight mission re-evaluation. The basic idea behind the model was to consider the messages flowing through the network as fluid quantities. Therefore, each node in a network can be modelled as a buffer, as shown in Figure 1.7 with maximum capacity $q_{max} \in \mathbb{R}^+$, where messages arrive at rate $\dot{a} \geq 0$ and depart the buffer at rate $\dot{d} \geq 0$.

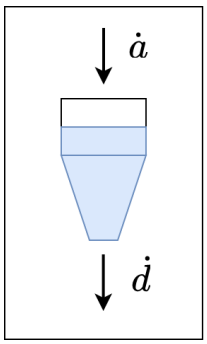


Figure 1.7: Representation of a Buffer Where a Fluid Arrives with Rate $\dot{a}$ and Departs with Rate $\dot{d}$

If $\dot{a} > \dot{d}$, the buffer can receive more messages as long as the current level does not exceed the buffer capacity. When they are equal, the buffer level will remain constant, and when $\dot{a} < \dot{d}$, the buffer will drain and, when completely empty, will eventually make the drainage rate equal to the arrival. To expand the idea of the buffer to a MANET, each node is represented as an individual *First In First Out (FIFO)* queue, as shown in Figure 1.8, so fluid departing a queue is just seen as arrival to the queue of downstream node. The messages are therefore propagated until they reach their final destination.

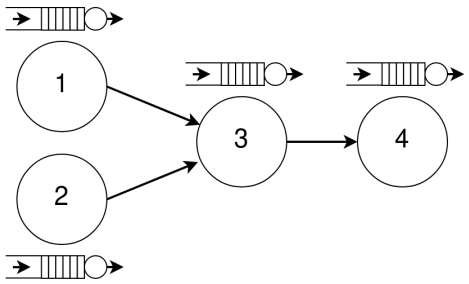


Figure 1.8: Representation of a MANET Using a Network of Connected Queues

The fluid model has been demonstrated to scale to large topologies and can be configured with bounded or unbounded network queues. We demonstrated how the model is useful to identify performance hotspots in topologies and how we could model more complex applications such as the Paxos consensus protocol. Additionally, we also studied control laws for swarms of mobile cyber-physical systems that can be used when deploying fully autonomous swarms of UASs.

1.3.2 Mobile Ad hoc Computing Emulator (MACE)

Another research interest, which is also related to methodology, was to experiment with existing algorithms and applications deployed in dynamic networks. Even though many simulators have been proposed, there was a gap in the state-of-the-art emulation tools for mobile ad hoc computing. Some important features were missing, such as a full-scale emulated deployment environment where prototypes coexist with off-the-shelf applications and more options for mobility control. We, therefore, propose *Mobile Ad hoc Computing Emulator (MACE)*, which will be described in Chapter 4 and was published as a conference paper [46]. MACE is a framework that enables the emulation of mobile distributed applications in a virtual environment so that the scenarios and topologies composed by mobile wireless nodes could be easily modified. This tool was crucial in overcoming the challenge of running algorithms in constrained devices before actually deploying in real hardware. It helps in reducing the reality gap in the early stages of research since it allows running sessions mixing software developed for embedded systems with RiotOS, QEMU virtual machines configured as constrained devices, QEMU virtual machines configured as multicore x86 computers and even Docker containers. It has also been used by other researchers to obtain interesting results, that at the moment of this document writing, have been accepted as a paper conference [28].

1.3.3 UTM Position Tracking and DDAA

This thesis also studies some real-world applications of the proposed models in the field of UTM. Even though UTM systems have advanced significantly in recent years, some aspects still need to be fully defined. In this thesis, we propose a distributed position tracking data layer for very low-level airspace, which was published as a conference paper [47]. We also have an ongoing research project that proposes an optimized offloading scheme that can be used in UTM DDAA systems. The project, which is part of a cooperation between ENAC

and TUD[10], has been accepted at a workshop [79]. Therefore, we present in this thesis the motivation, problem formulation, formalization of the optimization problem and some initial results using Monte Carlo optimization strategies.

1.4 Thesis Outline

This thesis is organized as follows. In Chapter 2 we describe more extensively the CPS swarms and the challenges related to information exchange in the context of distributed computing. Additionally, some important terminology that will be used throughout the document is also described, along with a description of the methodology used for the research. The remaining chapters are further divided into two parts. The first, with Chapter 3 describing a lightweight fluid model and how it was used to build a CPS swarm fast time simulator, and Chapter 4 describing how we designed an emulation tool used for experimentation. The second part focuses on how we applied the methodological tools to create some applications in the field of UTM. In Chapter 5, we describe the current state-of-the-art research on the UTM and propose a position-tracking system for very low-level airspace. In Chapter 6, we describe how an offloading algorithm can be leveraged for parallel computation of UAS collision avoidance algorithms in the context of UTM.

[10]Technical University of Dortmund

Chapter 2

Background and Methodology

Contents

2.1 Introduction

This chapter introduces some background knowledge that is useful throughout the rest of the document, and the overall application domains concerning the topics discussed. It defines the mobility and network models used in subsequent chapters, and establishes a connection between the CAP theorem and CPS swarms by describing the behaviour of network partitions, system availability and consistency in such deployments. Later on, it describes important aspects for the information exchange within the topology, by describing the concepts of algebraic connectivity, network density and network conductance. Finally, it describes the main challenges introduced by such domains, the methodology adopted to investigate and tackle such challenges.

2.2 Swarms of Constrained Mobile Nodes and Dynamic Networks

Swarms of CPSs are an emerging class of computational platform that has applications spanning several domains, such as industry 4.0, maritime navigation or aerial surveillance, earth observation, search and rescue operations and planetary exploration [92] [11] [146]. According to [120]:

Definition 2.1
CPS *- "Cyber-physical systems combine cyber capabilities with physical capabilities to solve problems that neither part could solve alone."*

And according to [19]:

Definition 2.2
Swarms *- "Complexity science has shown that collective behaviors in animal groups, that is swarms, emerge from repeated local interactions between neighboring individuals. It has also revealed that a set of simple local interaction rules applied to very simple artificial agents gives rise to complex patterns possessing long-range and long-lasting dynamic order."*

By adopting such a paradigm, swarms of CPSs can execute tasks in places and environments that otherwise would be inaccessible or inefficient for larger robots [21] or when the presence humans in hostile environments is frowned upon. They are also more resilient since the failure of one or some nodes does not hinder the continuation of the application carried out by the remaining nodes. However, such devices are usually constrained in processing power, communication capabilities and energy availability when compared to cloud servers or larger nodes for instance. Moreover, because the links and the robots themselves are subject to failures, the applications must be designed to tolerate at least benign failures, ensuring a higher level of availability and reliability when compared to a mission performed by only an isolated robot or by a centralized architecture. Such benign failures can also affect the overall performance of the system, due to added constraints to cope with them. Therefore, one of the challenges of such applications is the deployment planning to ensure high fault tolerance, whilst keeping high-performance levels. When considering the mobility of the nodes inevitable, bio-inspired post-deployment strategies are often employed in swarm control. And when considering swarms running distributed applications, implementing algorithms whose design goal is to maintain the nodes' capability to exchange data is imperative.

Some important factors need to be considered when engineering a swarm of CPSs. First, regarding the control, it can be centralized via a base/control station that sends individual commands to each node or decentralized. Centralized control has the advantage that considering the controller station has the full view of the swarm in real-time it can thus run controlling algorithms with high precision. The main disadvantage of such control is the introduction of a single point of failure and the requirement that the controlling station must be in a visual line of sight to all nodes or rely in some sort of relaying, which reduces the robustness and determinism. However, it is possible to eliminate the dependence on a controlling station by using decentralized control and control laws, where each node requires only

information regarding its neighbours or a pre-defined k-neighbourhood. Previous works have introduced ways to make such swarms resilient during the mission by proposing composed control laws that aim at maintaining the connectivity [136], increasing robustness [56] and avoiding collision between the nodes. Chapter 3 covers the theory and implementation of some of those decentralized control laws when applied to an application of CPS swarm.

Another important aspect is how the nodes exchange information within the swarm. When a data-intensive application is run on a swarm, the collected data needs to be processed, stored or forwarded to another architecture level. When the nodes are deployed as a MANET, data flows forward on a multi-hop way until it reaches its destination within the swarm. However, it is also often the case where the swarm communicates with another topology level, and the traffic is ultimately concentrated at the edge of the network. As examples of such concentration, there are edge and fog [96, 176, 39] applications or the bent pipe architecture sometimes employed in nano-satellite swarms [36]. In both cases, there is a high probability of bottleneck creation if traffic is left uncontrolled.

Depending on the type of swarm, the communication physical layer uses different types of wireless medium such as optical, acoustic or *Radio Frequency (RF)*, but the focus in this work will be on RF, and more specifically the IEEE 802.11 stack when using the emulator. More details about the network model adopted throughout this document can be found in Section 2.2.2.

2.2.1 Swarm Mobility

The source of dynamic behaviour on a network can arise due to constant change in the topology introduced by constant route changes, link failures or even creation of sporadic bottlenecks. The dynamic behaviour can even be observed on topologies where wireless nodes are static, or when they are connected via wired links, but their connection are extremely failure prone. However, in this thesis, we focus on networks which the source of dynamic behaviour is due to the mobility of the nodes. Nodes' mobility can be consequence of a non intentional behaviour, such as nodes deployed in a fluid. Examples are non propelled robots deployed in sewage systems, maritime streams, storms and etc. On another spectrum are the topologies created with SPPs, which are robots that have means of moving by decision of the nodes' internal control system or by an external centralized controller. We will focus on SPPs and model and explore mobility using academic mobility models, or robotic swarm motion control depending on the application or study in case.

Mobility Models

In the simulated environments for mobile networks, instead of detailing with precision the motion of the SPP in the environment by detailing physics models for thrust and friction, it is common in academic works to use approximation models. Such models directly update the nodes' Cartesian coordinates $\mathcal{P}(t) \in \mathbb{R}^3$. There are many of such models in the state-of-the-art research, for example, the authors of [114] listed the most common ones applied in FANETs. In these models, at each time step, the position is changed by a $\delta_p(t)$ based on a pre-defined velocity $\mathcal{V} \in \mathbb{R}^+$, with $\delta_p(t) \in \mathbb{R}^3$ being a set $[x, y, z]$. To calculate $\delta_p(t)$, each specific mobility model has its own equation, and the position $\mathcal{P}(t + \delta_t)$ is, hence, calculated

by the generic Equation 2.1.

$$\mathcal{P}(t+\delta_t) = [x(t)+\delta_{p_x}(t), y(t)+\delta_{p_y}(t), z(t)+\delta_{p_z}(t)] \tag{2.1}$$

Motion Control Models

In addition to the possibility of using mobility models as defined in the previous section, motion control models can be implemented for the nodes' mobility. This is particularly interesting because they can be controlled by the influence of different control laws, from which the outputs are velocity vectors. Such control laws can be used to enforce desired behaviours on the swarm [19] [130], which can be often decentralised. When using such models, each node's j position $\mathcal{P}_j(t) \in \mathbb{R}^M$ state is updated by an increment $\dot{\mathcal{P}}_j(t)$.

$$\dot{\mathcal{P}}_j(t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) \\ sin(\theta) \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w \tag{2.2}$$

therefore,

$$\mathcal{P}(t+\delta_t) = \mathcal{P}(t) + \dot{\mathcal{P}}_j(t)\delta_t \tag{2.3}$$

with $v \in \mathbb{R}$ being the velocity measured in m/s as the resultant of an applied force and $w \in \mathbb{R}$ the angular velocity measured in rad/s.

2.2.2 The Network Model

Unlike infrastructure-based wireless topologies, ad hoc networks do not rely on a base station to manage the connectivity among mobile nodes. These network topologies, which are composed by wireless mobile nodes, are modelled as an undirected graph $T = (N, E)$. The set of vertices N contains n nodes, where N_i, with $i \in \mathbb{N}$ and $0 \leq i < n$, and the set of edges E is composed by the edges $e = (N_j, N_k)$, where the distance between nodes n and k, $d_{jk} \leq d_r$, being d_r the communication range. The latter is defined by the wireless radio technology modelled, which allows a maximum bandwidth $B \in \mathbb{R}^+$ defined in Mbps. The radio models considered for this work were the *Unit Disk Radio* and the *Signal Decay with Log-Normal Shadowing*

The Unit Disk Radio Model

For simplicity, every node in the vicinity of one of the nodes participating in the communication graph T, is henceforth considered a participating node. To establish connectivity between the nodes, the Euclidean distance between them is calculated, and when $d_{jk} \leq d_r$, the nodes can communicate with a one-hop distance. When $d_{jk} > d_r$, the nodes can eventually still communicate with every other node in the same connected component via multi-hop routing.

The Signal Decay with Log-Normal Shadowing

In this model, many factors contribute to the signal decay from source to destination. The first factor to take in consideration is the path loss, show in Equation 2.4, which represents

the signal strength attenuation proportional to the distance. The constant α, the path loss exponent, defines the rate of signal decrease and depends on the environment [105].

$$\beta(d) = \alpha 10 log_{10} d \tag{2.4}$$

Additionally, the model takes in consideration also a stochastic attenuation modelled as a Gaussian distribution, X_σ, with mean value of zero as shown in Equation 2.5. This distribution models the attenuation added by flat fading.

$$X_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.5}$$

Taking the previous factors in consideration, the signal strength at the receiver $P_r(d)$ is lower than the transmitted P_t as defined by Equation 2.6.

$$P_r(d) = P_t - \beta(d) - X_\sigma \tag{2.6}$$

Since the nodes can be constantly moving, the distance between two nodes in line of sight is continuously changing. It is therefore expected that the variation in distance will change the strength of the connection between the nodes, thus changing the signal to noise ratio. The environmental noise depends on the deployment conditions, and it is taken in consideration for experimentation when choosing the value of the constant P_t.

As seen before with Equation 2.6, the receiving power decreases with the distance to the emitter. so considering a noise value $P_n \in \mathbb{R}$ measured in dB, the *Signal to Interference and Noise Ratio (SINR)* at the receiver is $SINR_r = P_r(dB) - P_n(dB)$. Decreasing the $SINR_r$ increases the *Packet Error Rate (PER)*, decreasing the actual bitrate available for communication. Using the Shannon Equation 2.7, it is possible to calculate the maximum possible bitrate with an available bandwidth frequency.

$$C = W log_2(1 + SINR_r) \tag{2.7}$$

where $W \in \mathbb{R}$ is the available bandwidth in MHz and $C \in \mathbb{R}$ the available bitrate in Mbps. Figure 2.1 shows an example of the bandwidth attenuation att according to the log-normal model compared to a simple exponential attenuation $att = e^{-d^2/2\sigma^2}$ with $\sigma = 70$.

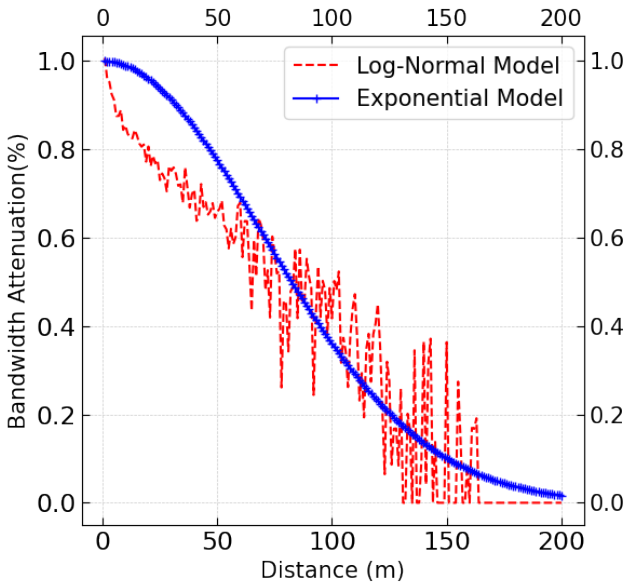


Figure 2.1: Normalized Shannon Bandwidth Dt = 15dB, α=3, W=20MHz, $P_n = -50dB$

The signal between nodes j and k decays proportionally to the communication gain $G_{jk}(t)$.

$$G_{jk}(t) = \frac{\mathcal{G}}{d_{jk}^2} \tag{2.8}$$

where $\mathcal{G}$ is a constant that represents the gain at one meter distance [172]. With the gain, it is possible to predict the loss of theoretical bandwidth leading to an actual value $\tilde{B}$.

$$\tilde{B_{jk}}(t) = B\, log_2 \left(1 + \frac{p_j(t) G_{jk}(t)}{\sigma^2}\right) \tag{2.9}$$

where $p_j(t)$ represents the transmission power at node j and time t [172].

Other Environmental Aspects

Among other aspects that influence negatively the performance of applications deployed in MANETs, one that has a big impact is the physical medium. In wired networks is relatively easier to introduce collision avoidance protocols for the network packets that allow many nodes to share the medium. Even though when using RF it is harder to know the correct time to transmit and avoid collision, many different MAC protocols have been developed. However, even with the more recent protocols, it is still not possible to achieve full use of the available radio bandwidth. Not only the communication between two nodes is affected, but considering a multi-hop communication path, the throughput is affected at each hop. To illustrate this aspect, we run an experiment with MACE where one node runs an *iperf3* client and all other nodes run servers. This way we were able to measure the throughput at increasing number of hops. Figure 2.2 shows the results of this measurements illustrating the exponential decay.

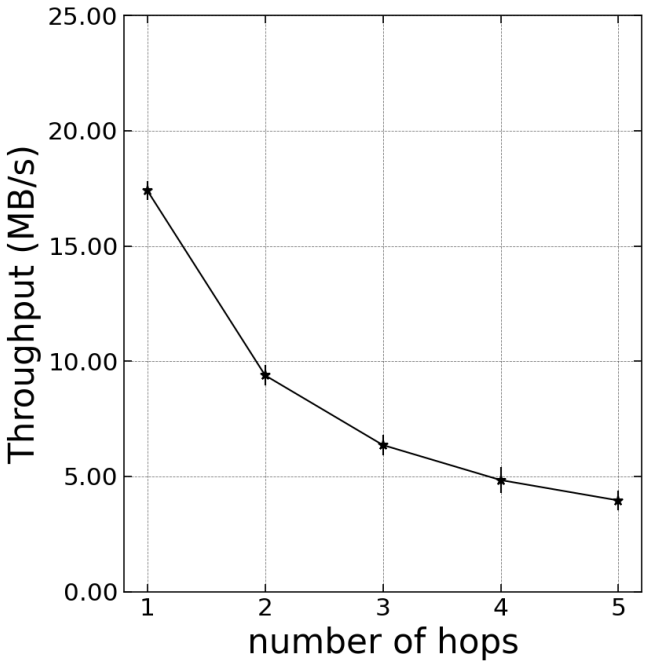


Figure 2.2: Measured Throughput with Traffic Injection Across Increasing Number of Hops Illustrating the Overhead Introduced by MAC Protocols

For this experiment the medium was simulated using *Extendable Mobile Ad hoc Network Emulator (EMANE)* and configured as IEEE 802.11g with theoretical bandwidth of 54*mbps*.

2.2.3 CAP Theorem and Swarms

In a swarm, due to the network's dynamic behaviour, it is expected that network partitions or transient communication failures as described in Section 2.2.6, will happen with a considerably high probability. The constant movement of the nodes and routing updates might not create necessarily a partition, but increase the delays between the nodes so that the effect on the algorithm can be similar to a transient network partition. The Sections 2.2.4 and 2.2.4 mention that in some situations, the availability, partition tolerance and consistency will vary depending on how the system is engineered and deployed. The trade-off between these properties has been extensively studied and was defined as the *Consistency Availability Partition (CAP)* theorem.

The CAP theorem [57], basically states that is not possible to achieve high availability, consistency and partition tolerance altogether. The system designer must accept a trade-off between the properties by relaxing them until the appropriate safety and performance levels are compatible with application's requirements. Each individual property of the theorem will be introduced in the following sections.

2.2.4 Consistency and Swarm Intelligence

Stating simply, consensus between the nodes is achieved when the client can read the same value, after a write operation or even the initial condition, regardless of which server i.e replica, it queries. The requirement for a strong or relaxed consistency depends ultimately on the application. For example, a distributed web cache does not require the same level of consistency as the UTM position tracking system described in Chapter 5, or a banking system. There different ways of achieving consensus, and now we introduce quorum-based and convergence consensus. Acording to [23]:

Definition 2.3
Consensus *- Techinique used by processes to agree on a common value out of initially proposed values.*

Also, to successfully achieve regular consensus, some safety properties must be guaranteed: termination, validity, integrity and agreement. They are defined in [23] as:

Definition 2.4
Termination *- "Every correct process eventually decides some value."*

Definition 2.5
Validity *- "If a process decides v, then v was proposed by some process."*

Definition 2.6
Integrity *- "No process decides twice."*

Definition 2.7
Agreement *- "No two correct processes decide differently."*

Quorum-based Consensus

The basic principle behind a quorum-based consensus is to reach an agreement by simple majority voting, as used in Lamport's Paxos [81] for example. In Paxos and other consensus protocols, the client requests to write or alter some data into a system, and the system needs to cooperate by means of voting to commit the operation across the replicas. Hence, any read operation performed after this commit will result in the same consistent value as long as there is enough quorum, and all participating processes are only susceptible to benign faults. When it is expected that some nodes might behave maliciously, different type of consensus protocol needs to be implemented, and the system availability might decrease because the number of nodes that can fail reduces to $f = 1/3N$). One characteristic of Paxos is the communication complexity, or the number of steps required to perform what would otherwise be realized by only one step. This, adds overhead to the communication, increasing latency between the write operation and the acknowledgement that the operation has been committed. Figure 2.3 illustrates the steps taken by the algorithm.

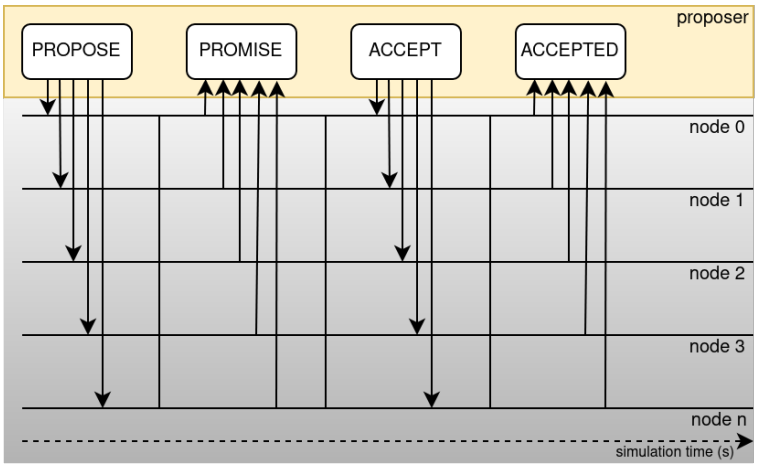


Figure 2.3: Main Steps of a Paxos Consensus

So, in the first proposal of the Paxos protocol, two round trips in the application layer are required to commit a write operation. If TCP/IP is being used as transport/network, that increases to four round trips, or more depending on the size of each message and maximum allowed size of the network packet. Paxos of course has evolved, and some variations require less steps in special circumstances. Those variations can reduce the communication latency even under high throughput. There are some aspects that need to be taken in consideration when running such an algorithm in dynamic networks. Due to constant link failures and constant change in routing paths some obstacles arise in establishing an expected maximum communication delay between the nodes. If the application is deployed on a TCP/IP stack for example, each packet exchange requires an ACK from the recipient, and when the link is broken before the ACK, the individual operation will be blocked until a timeout is reached. A link fault can be seem by the system the same was as node fault, and having many replicas allow that many link failures occur before the whole write operation is blocked waiting for a time-out, when it can finally be considered a failed write operation. Having a UDP can reduce the number of required *Round Trips (RTs)*, but impose the need to implement similar acknowledgement system on the application layer. Therefore, introducing dynamic behaviour in the network complicates the prediction of a upper bound in the communication delay. This renders an asynchronous network, in which consensus with fail-stop and malicious faults has been extensively studied [38, 49, 20], and although with reduced availability, reaching agreement is still possible. Hence, an algorithm deployed in such network would need to rely on time-outs that would ideally correlate to the average velocity of the nodes, so the more dynamic the network, the larger the timeouts.

Even though an algorithm such as a leader-based implementation of Paxos has high communication complexity when running an application that requires an intense exchange of information, for a swarm direction convergence or a rendez-vous decision only one commit would be needed. Figure 2.4 shows a waterfall plot of a multipaxos state machine replication running on the topology illustrated in Figure 2.9a, and executing five consecutive write operations with interval of two seconds between them. In this figure, time is discretized in 1 second buckets, so it should not be read as the representation of action duration, but as a heat map representing which nodes concentrated more traffic, which is a possible indication

of susceptibility to bottlenecks.

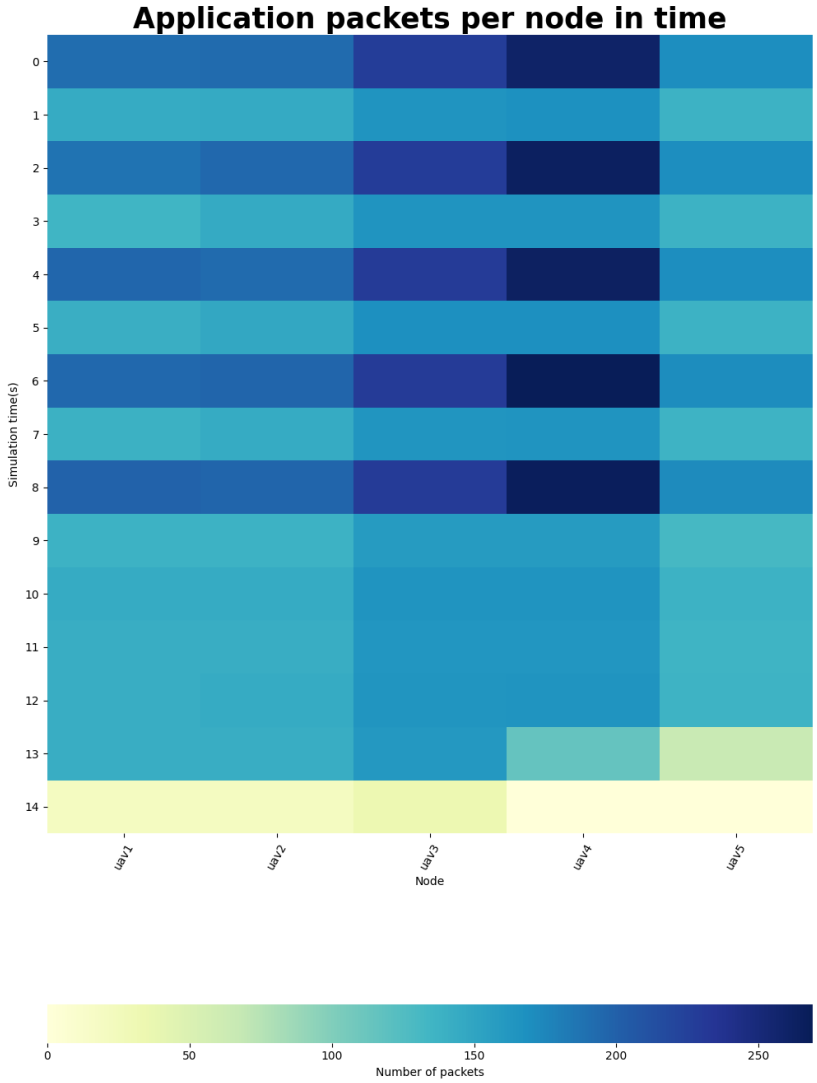


Figure 2.4: Waterfall Representation of Nodes Performing Quorum-based Consensus

As stated before, the network traffic between the consecutive write executions is very low because there is no need exchange of messages when in idle, besides what is used for fault detection and synchronization. Another characteristic of leader-based consensus, is the traffic concentration on the leader (node uav4), and in this specific case the node uav3, which serves as an obligatory path for traffic traversing the network. Latency wise, after running 50 consecutive write operations, the median latency for each write was 24.6ms. This experiment was run with under a realistic scenario with message exchange via TCP sockets and with the help of an emulator that will be further described in Section 2.4 and detailed in Chapter 4. On the other hand, a quorum-based consensus can be much faster to converge than a *swarm convergence consensus*.

Convergence Consensus

In dynamic networks, as in nature, it is possible for the nodes to only rely on their neighbours' state, therefore not requiring knowledge from the whole topology to guarantee correct function [60]. In such cases, as first observed in flocks of birds, nodes create a local cluster based on a degree of distance, such as k-neighbours. A convergence consensus can allow the participating processes or nodes to agree on a value, e.g. direction, velocity or rendezvous location, by gradually converging to a common value. This, therefore requires the exchange of information at several steps before convergence.

It has been observed in nature that the actual consensus interaction and the eventually required information exchange happens only between a local k-nearest neighbourhood, not the entire topology [19]. So, in a swarm of CPS, each node i would build a neighbourhood table N_i, with which they will exchange data, and each unsynchronized information exchange round, use Equation 2.10 to update its local state.

$$\dot{\theta}_i(t) = k \sum_{j \in N_i} a_{ij}(t)(\theta_j(t) - \theta_i(t)) \tag{2.10}$$

where $k \in \mathbb{R}^+$ is a configurable gain, $a_{ij}(t)$ is the adjacency matrix as defined in Section 2.3.1, and the referred state $\theta_j(t)$, can be for instance the direction or velocity of the nodes. Since the convergence is gradual, it is acceptable that some of the messages are lost. Not requiring strong consistency for each message exchange reduces the overhead in the application layer. Also, accepting spurious packet losses on the network layer, encourages the use UDP as transport protocol, which reduces the overhead in the network stack. On the other hand, what is made with one commit from a quorum-based consensus, require constant message exchange as can be seen in Figure 2.5. The figure illustrates a constant exchange of information during the whole emulation time, but with small overall size in bytes. Increasing the number of lost messages has an impact on the time to reach the consensus, possibly affecting the performance of the application, but allows the system to be more resilient to partitions and as a consequence increases its availability.

2.2.5 Availability in Dynamic Deployments

One of the main characteristics aimed at when deploying an application running on a CPS swarm is availability. Suppose for instance a swarm N running a distributed store application. When a client requests a read or write operation it expects an answer, and the ability to provide it, even if not necessarily correct, determines the availability. By having one node $n \in N$ with failure probability $p \in \mathbb{R}^+$ with $0 < p \leq 1$, it is natural to think that increasing the number of nodes increases the application's availability. Although, as illustrated in Figure 2.6, increasing the number of nodes increases the probability that at least one node is currently unavailable, the probability that all nodes or a majority of nodes are unavailable simultaneously, drops exponentially. A node may be unavailable due to an intrinsic benign fault, such as the node stopped working or the application crashed, or because of a network failure or partition, or due to malicious behaviour.

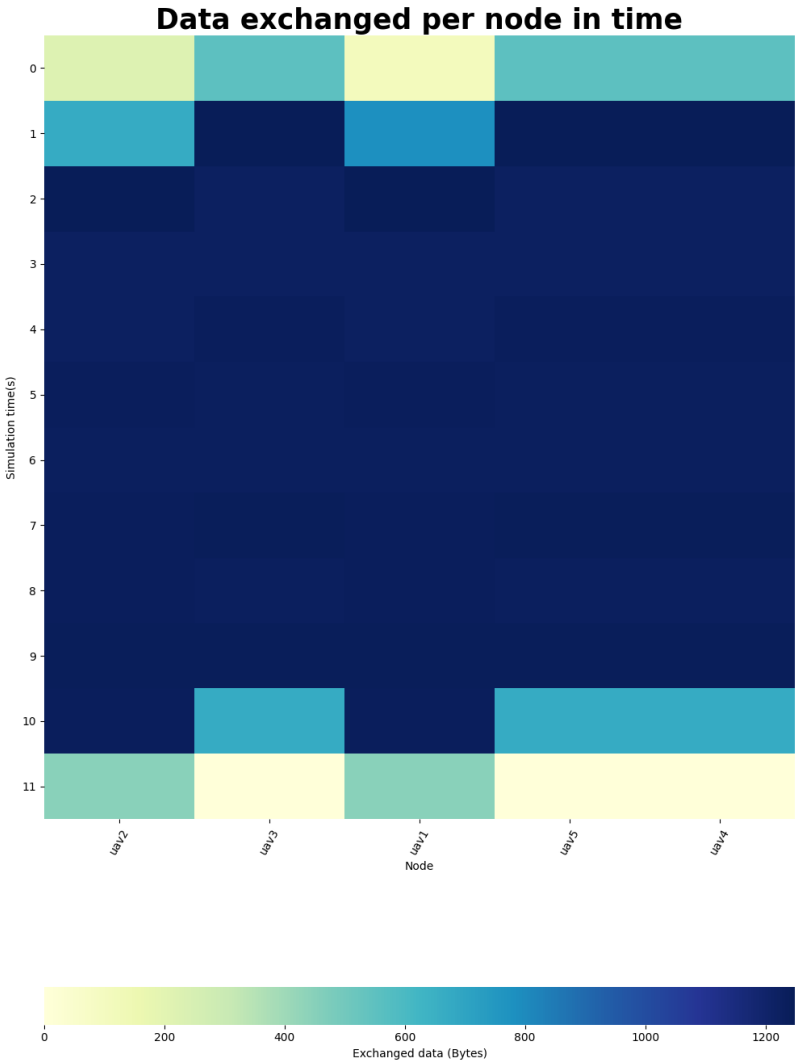


Figure 2.5: Waterfall Representation of Nodes Performing Convergence Consensus

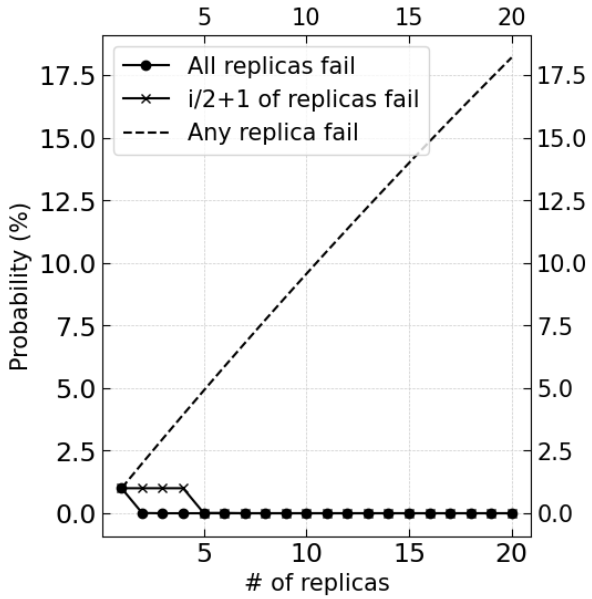


Figure 2.6: Probability of nodes failing with p=0.01

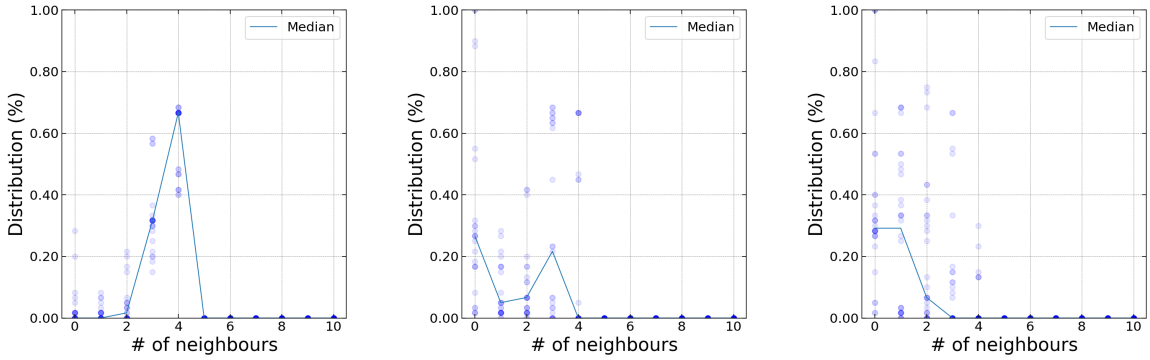
As mentioned before, availability does not imply that the answer provided to the client is correct. For stateful applications, after increasing the number of replicas, and consequently the availability, the need for consistency arise. It allows the client to receive from any available server, always the same correct answer. Stateless applications are much easier to scale and increase availability because there is not need for stored data consistency between replicas. There is, however, a desire of versioning consistency during deployment.

2.2.6 Dynamic Networks and Partitions

A partition occurs when a set of n nodes cannot communicate directly or via multi-hop routing. Then a partition can separate one node of the main graph or even divide the topology into two or more components. Deploying distributed applications in dynamic networks introduces new challenges, especially when trying to deploy an algorithm that requires knowing the state of the topology. It is for instance difficult to define the participating population, making it hard to guarantee *liveness* and *termination*. This is mostly due to the constant link failures and subsequent network partitions caused by nodes' mobility. Acording to [57]:

Definition 2.8
Liveness: "An algorithm is live if eventually something good happens. Availability is a classic liveness property: eventually, every request receives a response."

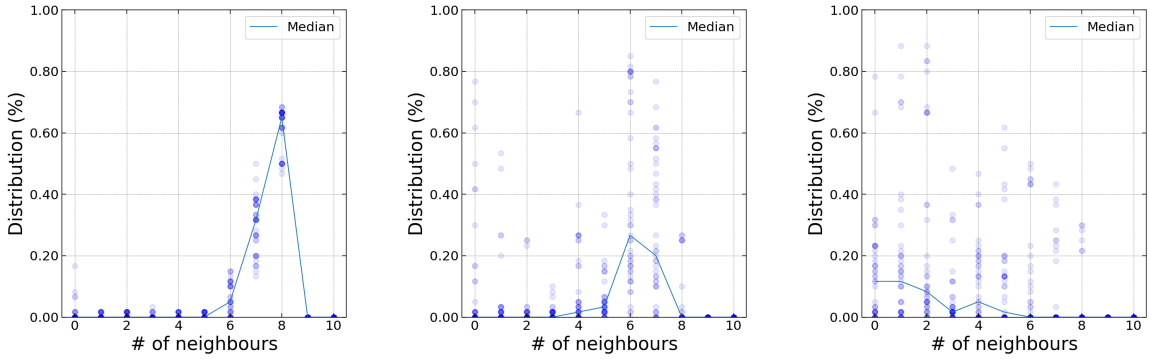
In large swarms observed in nature, it is difficult to imagine the possibility of global knowledge, such as needed when running a quorum-based algorithm. The same is also true when deploying a swarm of thousands of mobile autonomous systems since global knowledge would require information to be broadcast, which would significantly increase the network traffic. When dealing with dynamic topologies, the eventual transient network partitions and constant changes in topology render the network asynchronous. That means that is not possible to predict a maximum bound on the communication delay between any two participating nodes. A node could be disconnected for an arbitrary amount of time, so if the algorithm is waiting for this node's response, it will appear as temporarily stopped or blocked. We run an experiment in MACE to illustrate the effect that network dynamics can have in distributed algorithms. At each node a simple application broadcasts periodically every $500ms$ a *hello* message via *User Datagram Protocol (UDP)* sockets so that the other nodes can build a neighbour list. If a node does not hear from a neighbour for some time, it removes the node from the list. We ran the experiment with 5 nodes in Figure 2.7, or 9 nodes in Figure 2.8, deployed in an area with different sizes, hence changing the deployment density. The nodes move using the random waypoint mobility model, with a velocity randomly chosen between 1 and 2 m/s. The figures represent for how long the node had a specific number of k-hop neighbours reachable via the routing protocol during a $60s$ execution repeated 5 times.



(a) 300x300 5 nodes (b) 400x400 5 nodes (c) 500x500 5 nodes

Figure 2.7: Number of Detected Neighbours with Different Nework Density. Emulation with 5 nodes.

It is clear from Figures 2.7a and 2.8a that a higher density help the algorithms in keeping an updated view of the participant nodes by reducing the number of partitions. In the situations of Figures 2.7c and 2.8c, a quorum-based application would have its availability and liveness severely affected, which could be measured by the increase in latency and reduced throughput. Though, if the application allow relaxing the required consistency, the application could make progress despite the presence of partitions.



(a) 300x300 9 nodes (b) 400x400 9 nodes (c) 500x500 9 nodes

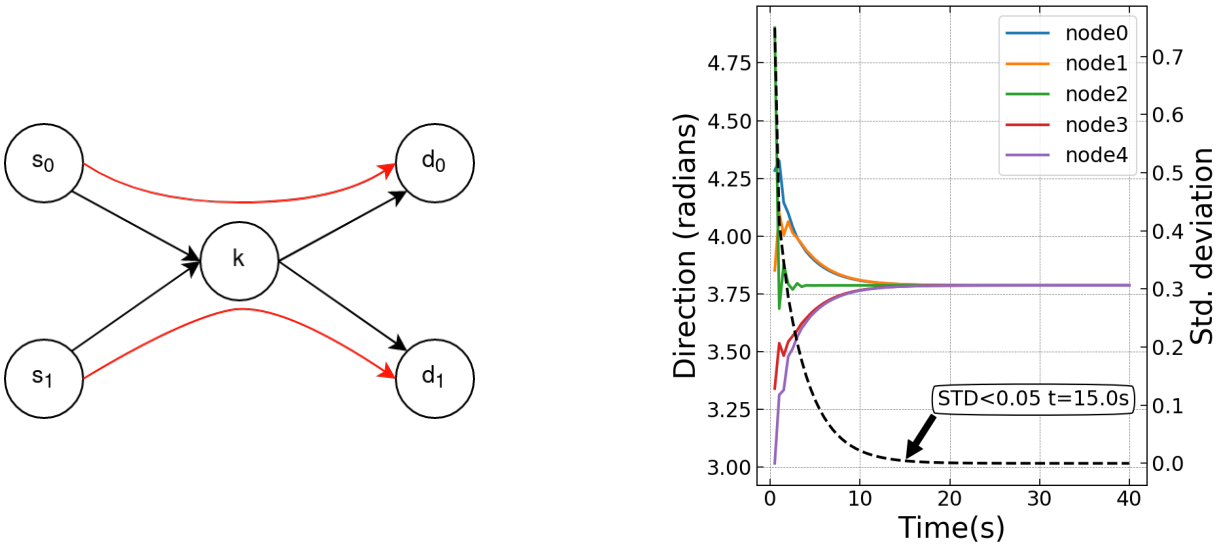
Figure 2.8: Number of Detected Neighbours with Different Nework Density. Emulation with 9 nodes.

2.2.7 Information Exchange and Swarms of CPS

In order to ensure *liveness* of most distributed algorithms, there is a need for information sharing among the peers. For instance, when a swarm, even in the biological sense, wants to converge in one direction, their current state must be shared with some of its neighbours. Such information transmission is therefore limited by the capacity these entities have to exchange information. Swarms of cyberphysical systems are usually connected via wireless links, as mentioned in Section 2.2.2, which are prone to connectivity instability and to throughput

degradation due to contention and radio interference in the medium [166]. Furthermore, the signal fades with distance and the throughput is affected at every communication hop [83], making it hard to establish upper bounds in the total communication latency. This asynchronicity of the network introduces challenges in running distributed algorithms that rely on quorums or algorithms that require real-time task scheduling.

It is expected that if the capacity of exchanging data by the topology is affected, also is the capacity of achieving swarm consensus, but how they are affected depend on each type of consensus. For example, let's use a simple example of *swarm convergence consensus* on a topology as shown in Figure 2.9a. Node k can directly communicate with all the other nodes and should converge faster to a common direction that will be followed by the others as shown in Figure 2.9b. In this example, the consensus introduced by Equation 2.10 was implemented with $k = 0.2$, and the MACE emulator was used for experimentation. The nodes exchange information by broadcasting UDP 1-hop packets with their current direction at a predefined interval of $t = 0.5s$.

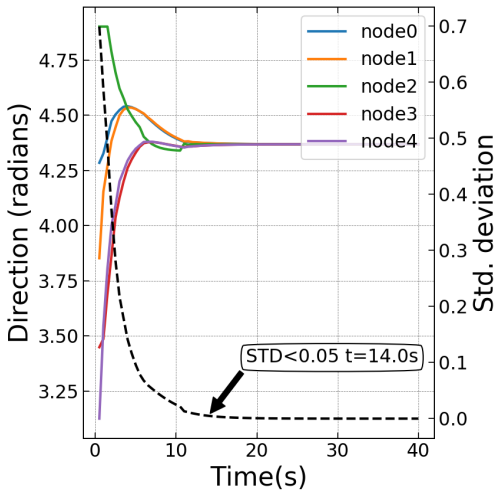


(a) Test Symmetrical Topology with 5 Nodes. k Node Experiences all Traffic Traversing the Network

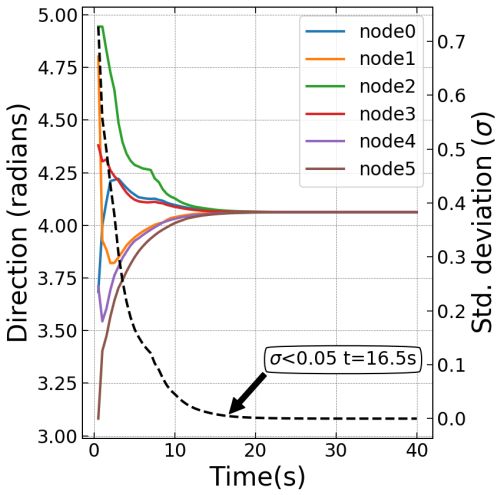
(b) Convergence with Low Traffic Being Injected

Figure 2.9: Swarm Direction Convergence Consensus Experiment with Low Injected Traffic

Intuitively, injecting a stress traffic from $s0$ and $s1$ towards $d0$ and $d1$ respectively, should hinder the information exchange capacity of the topology and affect the consensus algorithm. This is illustrated in Figure 2.10a where node k being a bottleneck, converges slower, resulting in a final direction different from when network traffic was low. This opens the question whether taming the traffic flow in the topology could actually yield different results. For instance, adding a new node to reduce the bottleneck formed in node k at sufficient distance to reduce contention, achieves the results shown in Figure 2.10b, with a consensus value less skewed.



(a) Convergence with Stress Traffic Being Injected from s0 and s1 Towards d0 and d1 Respectively



(b) Convergence with Stress Traffic but and Extra Node to Support node k to Transfer the Load

Figure 2.10: Swarm Direction Convergence Consensus Experiment with Stress Injected Traffic

2.3 Connectivity, Density and Conductance

Taking in consideration which type of connectivity provided by the chosen technology, the nodes' capability to exchange data can be affected by the distance between them and the ambient noise. Additionally, the connectivity graph can influence how the multi-hop routes are formed as mentioned in Section 2.2.2. Hence, the technology and the density of nodes affects the conductance [180] of data in the network.

2.3.1 Algebraic Connectivity

As mentioned in Section 2.2, information sharing is of ultimate importance for CPS swarms. Often the nodes need to agree in direction, rendezvous, coordinate tasks executions, or schedule activities. For the system to function properly consensus on those decisions is required, and reliable information sharing is important to facilitate consensus. Spectral graph analysis has been used to measure connectivity of a topology by calculating the eigenvalues of the Laplacian matrix that represents the topology [48, 173]. As shown below in Equation 2.11, the Laplacian $L = [l_{ij}] \in \mathcal{M}_n(\mathbb{R})$ is calculated with the adjacency and the degree matrices $A = [a_{ij}] \in \mathcal{M}_n(\mathbb{R})$ and $D = [d_{ij}] \in \mathcal{M}_n(\mathbb{R})$, $L := D - A$.

$$L = \begin{bmatrix} d_0 & 0 & \cdots & 0 \\ 0 & d_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{n-1} \end{bmatrix} - \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix} \tag{2.11}$$

The degree matrix $D = \text{diag}\{d_1, d_2, \ldots, d_n\}$ can be created for instance with $d_i = \sum_{j=1}^{n} a_{ij}$, and represents the actual number of nodes at one hop distance. The adjacency

matrix is created based in the connection between nodes. So, in the most simplistic form shown in Equation 2.12, $a_{j,k} \in [0, 1]$ indicates if a_j is in direct line of sight with a_k.

$$a_{jk} = \begin{cases} 1, & (j,k) \in E \\ 0, & (j,k) \notin E \end{cases} \tag{2.12}$$

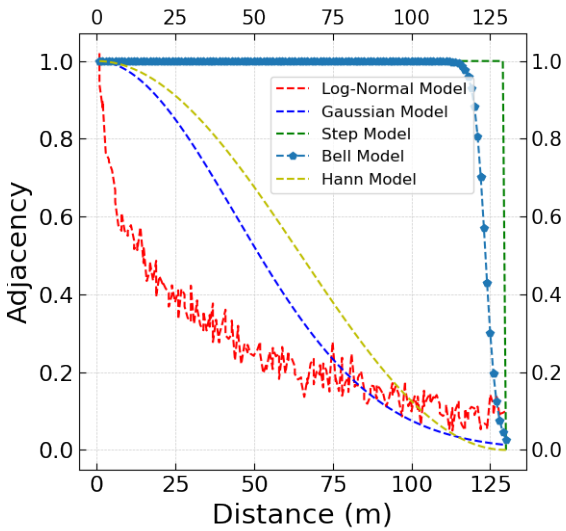
In this work however, since the nodes are connected by wireless interfaces, and in such cases the quality of the connection gradually degrades with the distance, other ways of defining adjacency can be used. One way that is commonly represented in literature is shown in Equation 2.13.

$$a_{jk} = \begin{cases} e^{-\frac{||p_k - p_j||^2}{2\sigma^2}}, & (j,k) \in E \\ 0, & (j,k) \notin E \end{cases} \tag{2.13}$$

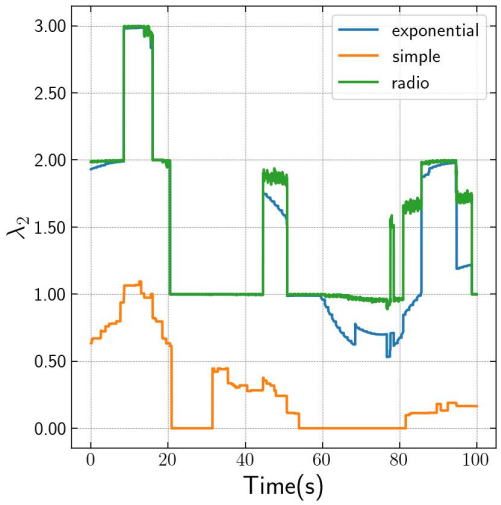
An alternative, is also to use the signal decay from Section 2.2.2 as shown in Equation 2.14.

$$a_{jk} = \begin{cases} P_t - \beta(||p_k - p_j||) - X_\sigma, & (j,k) \in E \\ 0, & (j,k) \notin E \end{cases} \tag{2.14}$$

Figure 2.11a shows the behaviour of the adjacency value with the distance for the three models previously introduced.



(a) Dt = 15dB, Range = 180m, α=3



(b) Dt = 15dB, Range = 180m, α=3, , $P_n = -55dB$

Figure 2.11: Algebraic Connectivity Experiment with Different Adjacency Models

In [48], the author showed that by calculating the second largest eigenvalue of the Laplacian matrix, it is possible to quantify a graph's current level of connectivity. In order to

compare the models before, a scenario with random walk mobility was replayed with the different ways of calculation the adjacency matrix and the result is shown in Figure 2.11b.

The step model has the advantage of making it easier to identify partitions, because with this model, when at least one node becomes disconnected, the connectivity goes to zero. Even though the log-normal model is more realistic, the Gaussian model introduces more gradual changes in connectivity value that might be interesting when deriving a control law that has as input the derivative of the connectivity. One way to enable information sharing is to ensure high connectivity in the topology. High connectivity also helps applications that require quorum and are affected by network partitions. Therefore, as will be shown in Chapter 3, the connectivity can be also used by a mobility controller that aim in keeping the measured connectivity of a swarm high.

2.3.2 Network Density and Conductance

Wired networks, benefit of much stable topology on deployed systems when compared to MANETs. Specially, when wireless nodes are sparsely distributed over an area, it is expected that depending on the amount of nodes, some set of nodes might not be reachable via a multi-hop route. The authors in [51] studied the correlation between the density of nodes and the connectivity to establish that on a topology with n nodes. By adding nodes to a deployment, the number of neighbours of each node also increases. Reaching a sufficient number of neighbours, converts a graph from disconnected to connected, and increasing this number enough can guarantee connectivity with probability of on. The study proved how to calculate these thresholds: each node should be connected with less than $0.074log(n)$ or more than $5.1774log(n)$ nearest neighbours, for the network to asymptotically disconnected or connected with probability of one, respectively. Figure 2.12 illustrates the k-nearest neighbourhood size according to the formula previously discussed.

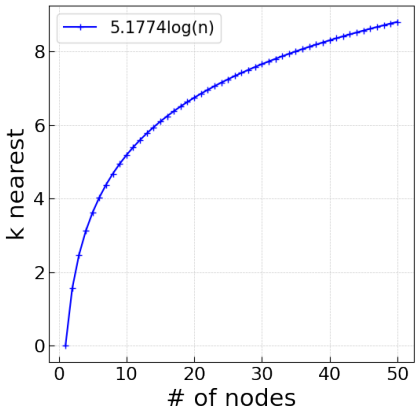


Figure 2.12: Connectivity Threshold with Increasing Number of Nodes

It is possible to verify experimentally this statement. We created a scenario with an area with fixed size, and increased the number of nodes to verify, with the increasing density of nodes, when the topology reaches a level of connectivity with probability close to 1.

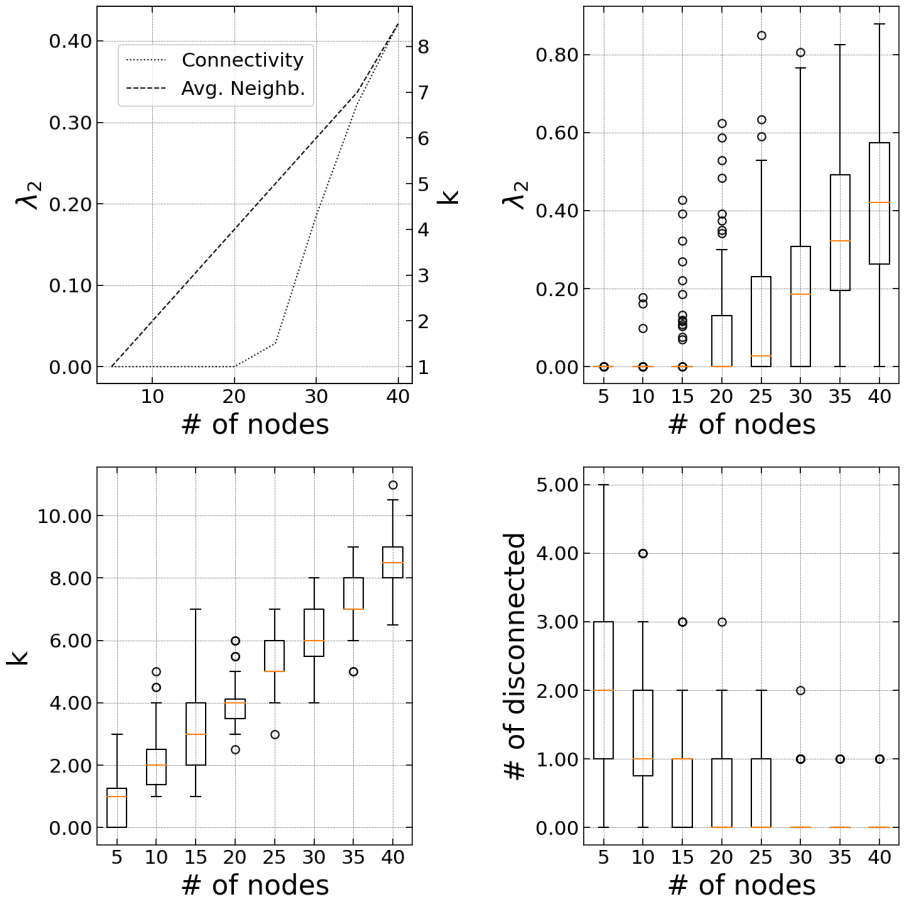


Figure 2.13: Influence of the Node Density on the Probability of Connectivity. Area: 400 m x 400 m

The results are shown in Figure 2.13, where it is noticeable that, for this area, the algebraic connectivity λ_2 starts to increase whilst the number of disconnected nodes start to decrease with around 35 nodes in the topology. According to Figure 2.12, with 35 nodes a k-nearest neighbourhood of 7.6 neighbours or more would yield a connected topology. With this number of nodes the size of the measured k nearest neighbourhood has a median of 7 nodes, value close to the predicted. Although usually perceived as having a negative impact on network connectivity, mobility can have a positive impact when exchanging information via gossip protocols in sparse networks [180]. When the mobility level is low, disconnected nodes or local clusters might never deliver messages being broadcast in other disconnected clusters. This situation is reduced when the mobility level is higher. All the results exposed so far were achieved by utilizing and emulator and a simulator developed during this project. The next section discuss the methodology used during the whole project in more details.

2.4 Methodology

In Chapter 1, a brief introduction to the problems we are aiming to understand and solve in this thesis was presented. First by describing the challenges introduced by multi-hop ad hoc networks, which are significantly increased with the introduction of *dynamicity* in the network. It thus incites curiosity not only to assess how previous solutions behave in such deployments, but also what could be done differently to overcome such challenges. Furthermore, we try to answer the question if could we create models and tools to support our own research and of others? Can we use these models and tools to study solutions that could potentially enhance the performance of distributed applications in dynamic networks?

In order to view and recognize such challenges, quantitative data had to be gathered. We aimed then at finding the correct tools and procedures to collect such data by using existing state of the art tools and models when possible, and to propose and develop our own when required. Most of the research presented here used data collected with realistic simulation scenarios based in existing research, and in some specific cases existing datasets that are identified and referenced. For each scenario, sufficient information is given to enable reproducibility of the experiments. The guidelines from [13] were followed as much as possible for each experiment performed with using emulations and simulations with a fluid model, as explained in the following subsections.

2.4.1 Emulation

For the first part of our research, there was the need to characterize the behaviour of some existing tools and algorithms with different scenarios that where not the ones aimed by the originating research. That and the eagerness to experiment with off the shelf applications pointed us in the direction of emulators, and more specifically emulators that allow controlling nodes' mobility and communication interfaces. After experimenting with existing tools, that culminated to the development of our own framework, which is properly defined in Chapter 4. The emulator generates realistic network data by creating *pcap* dumps that can be processed afterwards for extracting useful information, and extra traces that allow posterior analysis and reporting. Both dumps and traces are post processed with the help of Python scripts that automatically generate the figures presented in this thesis. Repetition of the experiments allow better statistical confidence of the collected data, but known negative effects of running emulations are also taken in consideration.

2.4.2 Fluid Model

Learning all the benefits and disadvantages from Emulators, we identified the opportunity to establish a network model, that would fill the gaps left by the emulator, such as the possibility or running fast simulations and better quality in the results regarding variability. A lightweight fluid model, properly described in Chapter 3, allowed us to run simulation sessions much faster and with reproducible collected data, with a level of realism and accuracy that was enough for the goals of studying the behaviour of swarm control and modelling applications. Even though the simulator was built on top of such model still function under discrete steps, the modelled message flow is treated as continuous quantities instead of discrete network

packets. The simulator is capable of generating detailed tracing information with the same resolution of the simulation session itself. It can create mobility traces that can be used to replay the simulation for observation or even re-running the same mobility pattern with different settings. All data has been also processed and plotted with Python scripts.

2.4.3 Metrics and Benchmarks

Both the emulator and fluid model allow the collection of several metrics useful for algorithm analysis. Most commonly used metrics for system analysis such as throughput and latency can be inferred by the individually collected metrics. The list below exemplifies some of such metrics, and a more comprehensive explanation on how they are derived can be found in Chapter 3.

- Throughput - It is possible at any point of the simulation, to know how much data is traversing each node per unit of time.
- Delivered data - It is also possible to know how much data was delivered at the final destination during the whole simulation.
- Dropped - When implementing the queues with bounded capacity, it is possible to know how much data was dropped during the execution.
- Queue level - Both total node queue level and the individual values depending on the destination are registered during the whole simulation.
- Queue latency - The average latency expected to be introduced by the queue based on Little's Law.
- Queue age - Average accumulative latency estimated at each node's queue.

By using these metrics, and implementing different ways of simulating the same scenario, it was possible to benchmark some solutions against each other. Additionally, by using the emulator, it was possible to benchmark also existing solutions and algorithms running on host or emulated hardware.

2.4.4 Discussion and Limitations

There are some aspects that must be taken into consideration when running an emulation[93], as it is not always needed. An emulation usually consumes resources of the host computer while trying to keep the real-time scenario, as it tries to keep up with wall-time. All the emulated resources and the applications being tested are sharing the same computer resources, and might in some cases be given less priority by the operating system. That creates challenges for the reproducibility of the collected metrics and usually requires that experiments are run more times, which is very time-consuming considering the scenarios run in wall-time.

Another important aspect was the difficulty to find academic datasets that reflected the scenarios tested in this work, which is a common problem in different *edge computing* scientific research fields [75], which is a contrast to the abundance of workloads and datasets available for *cloud computing* and *High Performance Computing (HPC)* [44, 163]. Therefore, in most

cases, synthetic data was used with the appropriate information on how to create the same data for reproduction. When academic datasets were used, appropriate references were added.

2.5 Conclusions

Cyber-Physical Systems are a class of devices combining computers and physical actuators. Combining several of them in a way that they can interact with each other to cooperate on a task defines a swarm of CPS. Such swarms increase mission robustness and allow reaching hard-to-access and distant places by running decentralized applications that require good balance in the network traffic to function with high performance. In this chapter, we introduced the connection between the CAP theorem and described it in the context of swarms by establishing a link between consistency and swarm intelligence, discussing how the availability of the nodes is impacted in dynamic deployments and the network partitions affect the swarm topology. We showed by experimentation how the dynamic networks impact the information exchange and how that can impact swarm intelligence algorithms. Furthermore, we showed experimentally that some main characteristics, such as network connectivity, density and conductance, are measurable and cause a direct impact on the data flow within the network. Those characteristics can be pragmatically exploited to enhance the application's performance, as will be shown in upcoming chapters. Finally, we discussed the methodology employed during the whole project, the rationale behind the development of the tools we proposed, the challenges faced during execution and limitations that need to be taken into consideration.

Part I

Modelling, Emulation and Simulation for Performance Evaluation

Chapter 3

Fluid Flow Model for Fast Simulations and Application Design in Cyber-Physical System (CPS) Swarms

Contents

3.1 Introduction

As mentioned in Chapter 2, a distributed computing system is a set of individual networked nodes that collaborate to execute a task or execute individual tasks that serve a common purpose. Such computing nodes can be static, as in traditional cloud infrastructures [176] or HPC centres. Otherwise, they can be mobile [170], which is a configuration more common in CPS, *Internet of the Things (IoT)* or *Mobile Edge Computing (MEC)*. When deployed as mobile nodes, new challenges arise to model, design and test distributed applications due to their dynamic nature and unreliable network connections. Such dynamic behaviour results in continuously-changing routes within the network, rendering the network traffic unpredictable under certain circumstances.

This chapter introduces a new lightweight analytical model to investigate the data flow in such dynamic architectures, and how certain topologies may hinder the overall system performance. As a tool, it complements the emulator that will be introduced in Chapter 4. One of the biggest challenges when developing simulation models is scalability. To tackle this challenge, we model the message passing as fluid quantities instead of discrete network packets. To simulate the receive, forward and deliver behaviours, the topology is modelled as a network of connected queues.

We implemented the fluid model in C++ and several other useful modules to simulate meaningful distributed applications. Previously, other works have been produced with similar goals, but yet, none modelled full dynamic topologies and could observe and record metrics during the whole time horizon. This chapter was also published as a conference paper [45].

3.1.1 Traffic Flow in Mobile Distributed Applications

As mentioned in [21] and [12], one of the first steps of modelling a swarm of CPS is to define the deployment topology or self-organizing models. It is, therefore, crucial to have the ability to search several different topologies and path-planning alternatives for better traffic flow, better coverage or both. A lightweight network model that can be employed as a faster-than-real-time simulator ads some conveniences. Mobile nodes have the intrinsic advantage of being able to move during runtime and thus rearrange the topology. By being able to quickly re-evaluate the traffic flow during a mission, the nodes can be potentially rearranged in real-time to improve the performance as done in [103] for connectivity maintenance. Topology control is a popular topic in *Wireless Sensor Networkss (WSNs)* [85, 33], and it is used as a way to increase network coverage or reduce energy consumption. While discrete network

models provide a higher granularity level, that is usually not required for many applications. Hence, models based on fluid mechanics have been proposed as a scalable alternative [154, 16, 133, 100]. Assuming a macroscopic point of view of the network flow, such fluid models do not keep granular information about each message. Therefore, they can simulate larger topologies and message flows faster, enabling fast mission planning and opening the possibility of its reevaluation during runtime.

3.1.2 Contributions

The main contribution of this work is the proposed lightweight, analytical model for distributed computing in mobile ad hoc networks. Such a model can be used to study swarm robotics deployments aimed at simulating network flows and finding events that can hinder the performance of mobile distributed systems. Due to its modularity, the model can potentially be extended to study routing or MAC protocols.

3.1.3 Chapter Outline

This chapter defines the proposed analytical fluid model in the following Section 3.2. The subsequent Section 3.3 defines a detailed system model based on existing state-of-the-art approaches that allows applications to be modelled and tested. The proposed model's accuracy is later validated in Section 3.4 in two ways:

- Modelling a data flow model and implementing on a network simulator (OMNet++), in MACE and in the proposed analytical model.

- Modelling the steady state performance of a bottleneck node and comparing the results with the fluid model proposed in [16].

The model is then used to verify the ability to extract valuable metrics for studying distributed systems, such as throughput and latency, by comparing stress loads on different topologies. Later, it is used to investigate the behaviour, during the whole simulation runtime, of factors that might hinder the performance of a specific topology by exposing transient queue levels and average latency in bottleneck nodes. Finally, the scalability of the model is assessed. The results are presented in Section 3.5 and Table 3.1 lists most of the notations used in this chapter to aid understanding.

Table 3.1: Notations

Symbol	Description
δt	Time-step
n	Number of nodes in the topology
d_{jk}	Distance between nodes j and k
d_r	Radio communication range
$\mathcal{R}$	Routing table
S_j	Set of nodes serving data flows for a particular node j
$O_{\mathcal{R}_j}$	Set of nodes being served as next hop for a particular node j
t_{mp}	Medium propagation latency
t_s	Serialization latency
t_{od}	End-to-end latency
$\mathcal{P}$	Position of a node
$\mathcal{A}$	Attraction position of a node
δ_p	Position variation for a node
$\mathcal{V}$	Velocity of a node
$\mathcal{L}_a$	Limiting area for a mobility model
λ	Flow creation frequency
α	Age of a message
p	Position of a message
$\mathcal{Q}_j^k$	Queue at $p = j$ for messages with destination $e = k$
$\mathcal{D}_j^k$	Drainage of queue at $p = j$ for messages with destination $e = k$
a_j^k	Arrival of queue at $p = j$ for messages with destination $e = k$
f	Average flow size
ρ	System load

3.2 Analytical Fluid Model

In this section, we introduce a fluid model that accurately models the message passing in the network. The model can be implemented together with the models described in Section 3.3 to build a simulator. In this model, each node in the network is abstracted as a queue, and messages are abstracted as fluid quantities flowing through them.

3.2.1 Messages

In order to model the flow through the topology, each transferred message is abstracted as $\mathcal{M}(\alpha(t), p(t), e)$. Each abstracted message has an accumulated travel delay, henceforth called age $\alpha(t)$ with $\alpha \in \mathbb{R}^+$, which will be further detailed in Section 3.2.3; a position $p(t)$ with $p \in \mathbb{N}, 0 \leqslant p < n$, that represents in which node the message is currently located; and the end node e with $e \in \mathbb{N}, 0 \leqslant e < n$.

3.2.2 Message Queues

As mentioned before in Section 1.3, we abstract the ad hoc network as a network of connected queues. However, since each message currently located at a node can go to a different downstream next hop, each node is abstracted as a set of queues. When a message flows from one node to another, they are placed in an output buffer as a corresponding amount of data. At each node, the queues evolve during execution, and data leaves the queues at a rate according to the currently available bandwidth connecting the two nodes at time t. So, as shown in Figure 3.1, each node queue is abstracted as N queues, each representing a final destination.

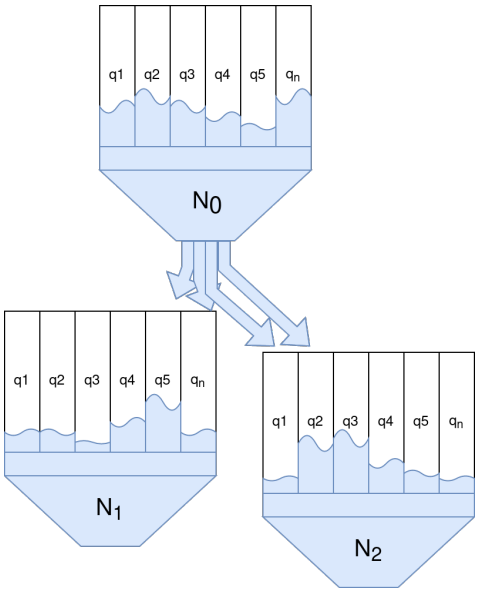


Figure 3.1: Multi Queue System Abstracting Each Possible Destination on the Topology

When an amount of data flows from node i to node k using node j as a route, this amount will be removed from queue Q_i^k and added at queue Q_j^k obeying the available departure rate. Therefore, each queue $\mathcal{Q}$ is a set with $|\mathcal{Q}| = n$, and each element represents a destination node $\mathcal{Q}_j = \{\mathcal{Q}_j^0, \mathcal{Q}_j^1, \mathcal{Q}_j^2, ..., \mathcal{Q}_j^{n-1}\}$, and the total queue level at a node j given by Equation 3.1:

$$\mathcal{Q}_j(t) = \sum_{i=0}^{n-1} \mathcal{Q}_j^i(t) \text{ with } p(t) = j \text{ and } e \neq j \tag{3.1}$$

When the level of more than one destination queue of a node n is higher than zero, they have to share the available bandwidth. Therefore, when a message is transmitted, it is added to a queue that drains at its maximum rate $\widetilde{B}_{jk}(t)$, with $\widetilde{B}_{jk} \in \mathbb{R}^+$, which is the available bandwidth between j and k. Since the system is discretized with time increments δt, $\widetilde{\mathcal{B}}_{jk} = \widetilde{B}_{jk}\delta t$.

Assumption 1. *Every destination is equiprobable. No model in place favours a chosen destination or that skew the ratio between ingress and egress.*

The individual queue drainage at j for messages with $e = k$ is therefore given by:

$$\dot{\mathcal{D}}_j^k(t) = \begin{cases} -\dfrac{\mathcal{Q}_j^k(t)}{\mathcal{Q}_j(t)}\widetilde{B}_{jk}(t), & \text{if } \mathcal{Q}_j^k(t) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

Since the destinations are equiprobable, there will be more messages served from larger destination queues; hence the term $-(\mathcal{Q}_j^k(t)/\mathcal{Q}_j(t))\widetilde{B}_{jk}(t)$. Each queue increases if flow arrives from other nodes with a rate higher than the possible outgoing flow. The arrival at j's queue with destination k, i.e. j is a routing node from a node i towards k, is defined by Equation 3.3.

$$\dot{a}_j^k(t) = \begin{cases} \sum_{\substack{i=0 \\ i \neq j}}^{n-1} \frac{\mathcal{Q}_i^j(t)}{\mathcal{Q}_i(t)} \widetilde{B}_{ij}(t), & \text{if } \mathcal{Q}_i^j(t) > 0; \mathcal{R}_{i,k}(t) = j \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

If the receiving node is not the final destination, its queue will receive a positive arrival flow with the same rate as the previous hops send. Therefore, each queue level derivative is:

$$\begin{aligned} \dot{\mathcal{Q}}_j^k(t) = & -\frac{\mathcal{Q}_j^k(t)}{\mathcal{Q}_j(t)} \widetilde{B}_{jk}(t) \mathbb{1}_{\mathcal{Q}_j^k(t)>0} \mathbb{1}_{j \neq k} + \\ & + \sum_{\substack{i=0 \\ i \neq j}}^{n-1} \frac{\mathcal{Q}_i^k(t)}{\mathcal{Q}_i(t)} \widetilde{B}_{jk}(t) \mathbb{1}_{\mathcal{Q}_i^j(t)>0} \mathbb{1}_{\mathcal{R}_{i,k}(t)=j} \end{aligned} \tag{3.4}$$

By observing Equation 3.4 for the case when $j = k$, i.e. data arriving at the destination node, the rate change of the queue can be only $\dot{\mathcal{Q}}_j^k(t) \geq 0$. At the end of the simulation, the value of $\mathcal{Q}_j^k(t_h)$ when $j = k$ will be the total data received at node j. These equations permit observing the queues' dynamics during the simulation execution.

3.2.3 Modelling Average Dynamic Communication Latency

As mentioned in Section 3.2.2, each message $\mathcal{M}$ has an associated parameter $\alpha(t)$, which represents the evolution of the message's age. The age, an abstraction for the accumulated delay since the message creation, is incremented every time the latter is being transferred or waiting in queue before arriving at the final destination (i.e. $\dot{\alpha}(t) = 1$). Since messages are not simulated individually in the proposed model, the end-to-end delay of each individual message is not measurable. One way of inferring an average value of the latency is to calculate the average accumulated delay of the fluid. For that, we modelled it by using the conservation of the first moment.

Assumption 2. *The MANET topology is a closed system. All information is created and consumed by the participating nodes.*

The primary rationale behind it is that the first moment, abstracted as αQ, queue level representing the density of messages at each node, and age representing the accumulated latency, is conserved when the delayed fluid flows from one node to another [1]. When messages are queued at a node, they age, i.e. they start accumulating the delay until they are served and transferred downstream. Thus, it can be inferred at any moment what is the actual expected end-to-end delay taking into consideration not only the current queue levels in each node, but also the accumulated latency from previous hops. For each individual queue $\mathcal{Q}_i^k$, there is an associated age α_i^k, where $\alpha \in \mathbb{R}^+$, $0 < \alpha < t_h$, and the evolution of the age in each queue is modelled by:

[1] This concept is adapted from [34], where instead of queue level, conservation equations where described for tasks moving from one data center to another.

$$[(\alpha_j^k(t-\delta t)+\delta t)(\mathcal{Q}_j^k(t-\delta t)-d_j^k(t-\delta t))+$$
$$+\sum_{\substack{i=0\\ i\neq j\\ \mathcal{R}_i^k(t)=j}}^{n-1}(\alpha_i^k(t-\delta t)+\delta t)d_i^k(t-\delta t)]-\alpha_j^k(t)\mathcal{Q}_j^k(t)=0 \tag{3.5}$$

Since, besides the current age, all the elements of the equation are known, $\alpha_j^k(t)$ can be isolated and calculated at each time step.

3.2.4 Losses and Retransmissions

Any wireless network data transmission can experience losses and delays. On a *Transmission Control Protocol (TCP)* connection, every packet transmitted needs an acknowledgement that it has been received, otherwise, the packet needs to be retransmitted. On a UDP connection, no acknowledgement is required, and when the application cannot bear the loss of packets, the request for retransmission needs to be issued by the application layer. Hence, usually UDP is preferred by applications that can cope with some level of packet losses. In order to simulate the loss of packets as mentioned in [100], data must leave the queue with a configurable probability of loss $p(t)$, with $p \in \mathbb{R}^+$ and $0 \leq p \leq 1$. Therefore, equation 3.4 can be simplified as:

$$\dot{\mathcal{Q}}_j^k(t) = d_j^k(t)(1-p(t)) + \sum_{\substack{i=0\\ i\neq j}}^{n-1} a_i(t) \tag{3.6}$$

So, when losses occur, the data will be deducted from the sending queue level but will not be added to the next hop queue. In wireless networks, the loss of packets most commonly occurs due to collisions in the physical medium or hardware error. However, packets can also be intentionally dropped when the receiving node has its buffer full or as a way to control traffic on the network to avoid congestion.

Drop Tail

In the drop tail model, data flows through a node n by using its FIFO queue Q_n as a temporary buffer before forwarding packets to the next hop towards the final destination. When the queues are bounded with a $q_{max} \in \mathbb{R}^+$ value, the level of Q_n increases according to the network dynamics until it reaches q_{max}. At this point, the queue rejects new packets that are henceforth dropped. So, at any moment in time, the space available q_a for receiving data from each node in the set of active source nodes S_n is defined by Equation 3.7.

$$q_a(t)_n = \frac{q_{max} - \mathcal{Q}_n}{|S_n(t)|} \tag{3.7}$$

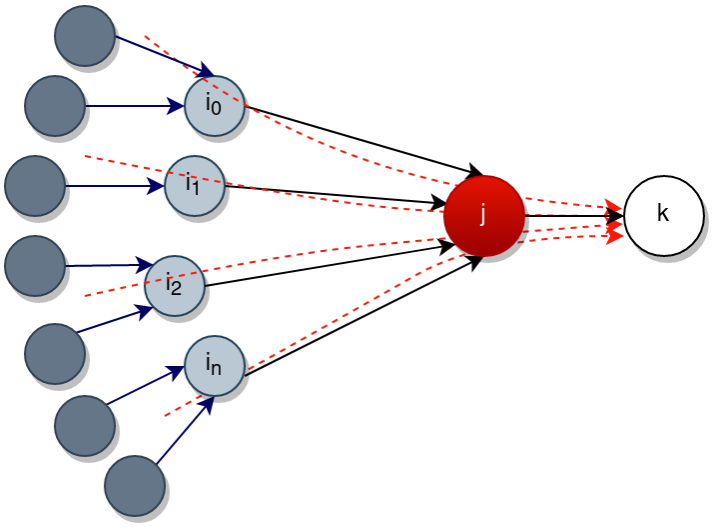


Figure 3.2: Representation of a Bottleneck Node Being Fed by Multiple Sources.

Therefore, considering the bottleneck node j with n sources i injecting flows towards k represented in Figure 3.2, the flow of packets leaving each source node in S_j is defined by Equation 3.8.

$$\dot{\mathcal{D}}_{ij}(t) = \begin{cases} -\dfrac{\mathcal{Q}_i^j(t)}{\mathcal{Q}_i(t)}\widetilde{B}_{ij}(t), & \text{if } \mathcal{Q}_i^j(t) > 0, \dot{\mathcal{D}}_{ij}(t) < q_a(t)_j \\ -q_a(t)_j, & \text{if } \mathcal{Q}_i^j(t) > 0, \dot{\mathcal{D}}_{ij}(t) > q_a(t)_j \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

Hence, the effective flows that are actually transferred to downstream nodes can be smaller than what leaves the source nodes.

Retransmissions

To allow collecting packet drop metrics and simultaneously model the possible retransmissions initiated by the application layer, the dropped quantities $\dot{\mathcal{D}}_d$ can be calculated by Equation 3.9.

$$\dot{\mathcal{D}}_{ij}(t)_d = \frac{\mathcal{Q}_i^j(t)}{\mathcal{Q}_i(t)}\widetilde{B}_{ij}(t) - q_a(t)_j \tag{3.9}$$

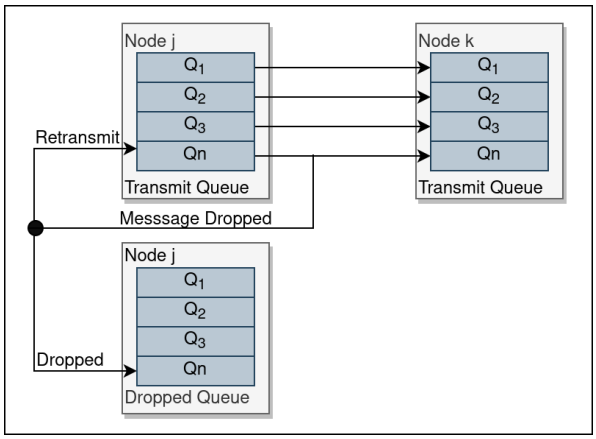


Figure 3.3: Illustration of the Transmit and Dropped Queues on each Node n

As illustrated in Figure 3.3, these quantities are added back to the source nodes' outgoing queues for retransmission and saved at a separate drop queue $\mathcal{Q}_d$ for metrics collection.

3.3 Mobile Ad Hoc System Model

This section introduces a model for mobile distributed systems by modelling its auxiliary subcomponents: the network, routing protocol, latency, mobility and message flow models. The models discussed in this section are needed to implement the fluid model proposed in Section 3.2 as a mobile distributed system simulator.

3.3.1 The Network Model

Unlike infrastructure-based wireless topologies, ad hoc networks do not rely on a central entity to control connectivity, and nodes are connected when within radio reach from each other. These network topologies, composed of wireless mobile nodes, are modelled as an undirected graph $T = (V, E)$. The set of vertices N is composed by n nodes, where N_i, with $i \in \mathbb{N}$ and $0 \leq i < n$, and the set of edges E is composed by the edges $e = (N_j, N_k)$, where the distance $d_{jk} \leq d_r$, being d_r the communication range. The latter is defined by the wireless radio technology modelled, which allows a maximum bandwidth $B \in \mathbb{R}^+$ defined in Mbps.

For simplicity, there is no membership control, and every node in the vicinity of one of the nodes participating in the connected graph T, is henceforth considered a participating node. In order to establish connectivity between the nodes, the Euclidean distance between them is calculated, and when $d_{jk} \leq d_r$, the nodes can communicate with a one-hop distance. When $d_{jk} > d_r$, the nodes can potentially still communicate with every other node in the same connected component via multi-hop routing.

3.3.2 The Multi-Hop Routing Model

There are many routing protocols applicable to mobile ad hoc systems [114]. For this study, a proactive, greedy, location-based routing protocol was modelled inspired by [52], since all the required metrics are simulated. Given the number of nodes n participating in the topology,

with $n \in \mathbb{Z}$, the routing table $\mathcal{R}$ is a matrix with dimensions $n \times n$. The line index i of $\mathcal{R}$ represents a node in the topology, while each column j represents a destination node from i, $\forall(i,j) < n$. Therefore, the value of $\mathcal{R}_{i,j}$ represents the next hop data currently in i with destination j. In this model, to update the routing table, the system periodically evaluates the current position of the nodes. It runs the Djikstra algorithm [37] to calculate the shortest path for each origin/destination pair. The table is therefore updated with the next hop towards every destination that yields the shortest path.

3.3.3 Bandwidth Sharing

The bandwidth provided by the MAC/PHY technologies of a wireless stack is considered to be fully available for a node. However, considering a technology where all connections share the same channel such as some versions of IEEE802.11, this system bandwidth is shared among the sources injecting data into the node and all the other nodes receiving data from the former. Considering the set of nodes serving data flows to node j, S_j, and the set of nodes $O_{\mathcal{R}}$ being served as next hop $\mathcal{R}_{j,n}$ for node j with n destinations, the bandwidth available for the link between nodes j and k as seen at node j is:

$$\tilde{B_{jk}}(t) = \frac{B}{|S_j(t)| + |O_{\mathcal{R}}(t)|} \tag{3.10}$$

However, the receiving node k will eventually be served by other nodes and serve other nodes as well. So, the actual bandwidth available $\tilde{B_{jk}}$ might be lower than expected by only observing j; i.e., every link will be limited by the lowest available bandwidth between two nodes. Considering any destinations n being served by k:

$$\tilde{B_{jk}}(t) = \begin{cases} \tilde{B_{jk}}(t), & \text{if } \tilde{B_{kn}}(t) > \tilde{B_{jk}}(t) \\ \tilde{B_{kn}}(t), & \text{otherwise} \end{cases} \tag{3.11}$$

3.3.4 Latency Model

There are two main drivers for delays in a distributed system: communication and computation. This thesis focuses on the former, and as expected on a multi-hop communication scheme, this has an impact on the overall network latency due to an accumulation of factors that add delays at each hop:

- Medium Propagation - Delay added due to the propagation of information in the medium. In this work, radio waves in the air are considered and their velocity approximated as $C = 299.792.458$ m/s, so the latency $t_{mp} \in \mathbb{R}^+$ is therefore $t_{mp}(t)_{jk} = d_{jk}/C$.
- Serialization - Delay added for actually transferring serialized data. Considering the current available bandwidth between the nodes $\tilde{B_{jk}}(t) \in \mathbb{R}^+$ measured in *Mbps* and the amount of data to be transferred $\mathcal{D} \in \mathbb{R}^+$ measured in *MiB*, the latency $t_s \in \mathbb{R}^+$ is therefore $t_s(t)_{jk} = \dfrac{\mathcal{D}}{(\tilde{B_{jk}}(t)/8)}$.

The total end-to-end expected latency from origin(o) to destination(d) $t_{od}(t) \in \mathbb{R}^+$ is therefore:

$$t_{od}(t) = \sum_{\substack{j_0=o \\ j_n=\mathcal{R}_{j,k}^{n-1}(t) \\ k_n=\mathcal{R}_{j,k}^{n}(t)}}^{k_n=d} t_{mp}(t)_{j_n k_n} + t_s(t)_{j_n k_n} \tag{3.12}$$

Additional delays added by hardware computation and due to medium access layers are not considered since they are dependent on specific hardware and chosen MAC protocol. The end-to-end latency can be estimated based on the path followed, and each hop's delays estimations [87, 111].

The latency increment added due to an increased distance between two nodes is caused by the latency induced by the bigger distance that must be covered by the electromagnetic wave and the reduced bandwidth introduced by the radio signal decay. The former is negligible compared to the latter and will not be considered. Pathloss can be easily modelled, but that concerns only the signal power decay, which is not necessarily connected with lower throughput.

3.3.5 Data-Flow Models

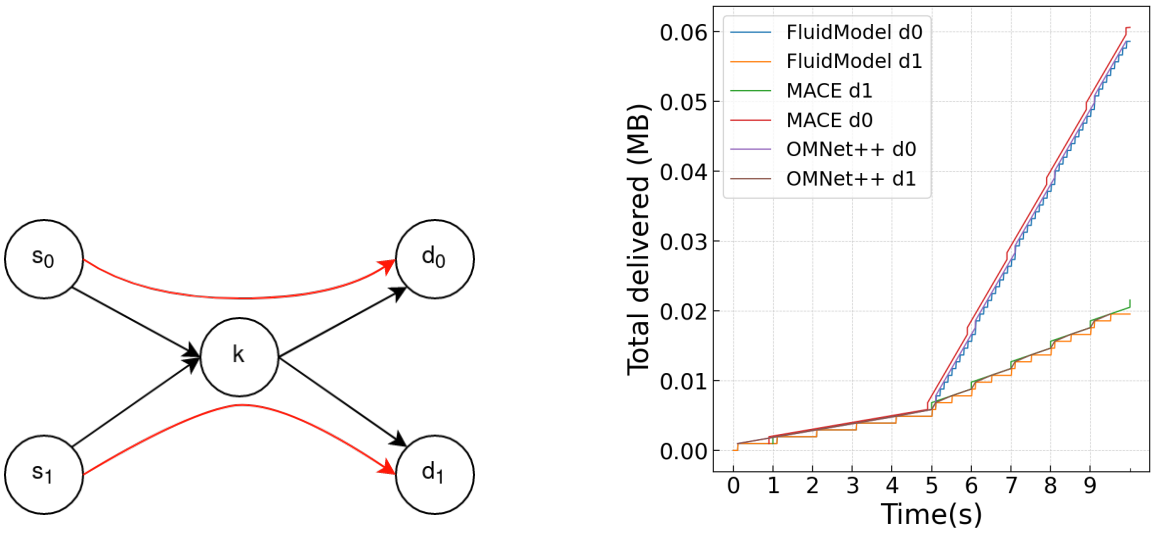
This data-flow model describes how the generated messages are distributed among the participating nodes during runtime. It mainly simulates the behaviour of messages being created by applications in one node and transferred to other nodes participating in the algorithm of the distributed application.

- Round Robin - each node becomes the source of traffic per round, and the source node is changed with every pre-defined service period defined by λ.
- Stochastic - the source node is selected randomly at each round according to a Poisson distribution. The flows are served following a pre-defined λ.
- Fixed - in this model, the source node (selected based on experiment objectives) is fixed during all the execution, and the destination is defined by a set of nodes. When left empty, all nodes will serve as destination. The data flows are served following a pre-defined λ.

3.4 Model Validation

To validate the behaviour of the proposed fluid model, a scenario where two source nodes inject data flows towards two different destination nodes via a bottleneck, as shown in Figure 3.4a, was simulated with the model, a simulator (OMNet++) and the MACE emulator. First, periodic flows of 1kB with arrival rate $\lambda = 1$ from S_0 to D_0 and from S_1 to D_1 are injected. After 5 seconds, a second periodic flow of the same size but arrival rates $\lambda = 0.1$ from S_0 to D_0 and $\lambda = 0.5$ from S_1 to D_1 is introduced.

Figure 3.4b shows the evolution of the total delivered data for each tool and destination node. It shows that even having fluid quantities that mix in the bottleneck node, it is possible to achieve results with the same precision as a discrete simulator without the need to



(a) Topology

(b) Results

Figure 3.4: Model validation with a network emulator and a simulator

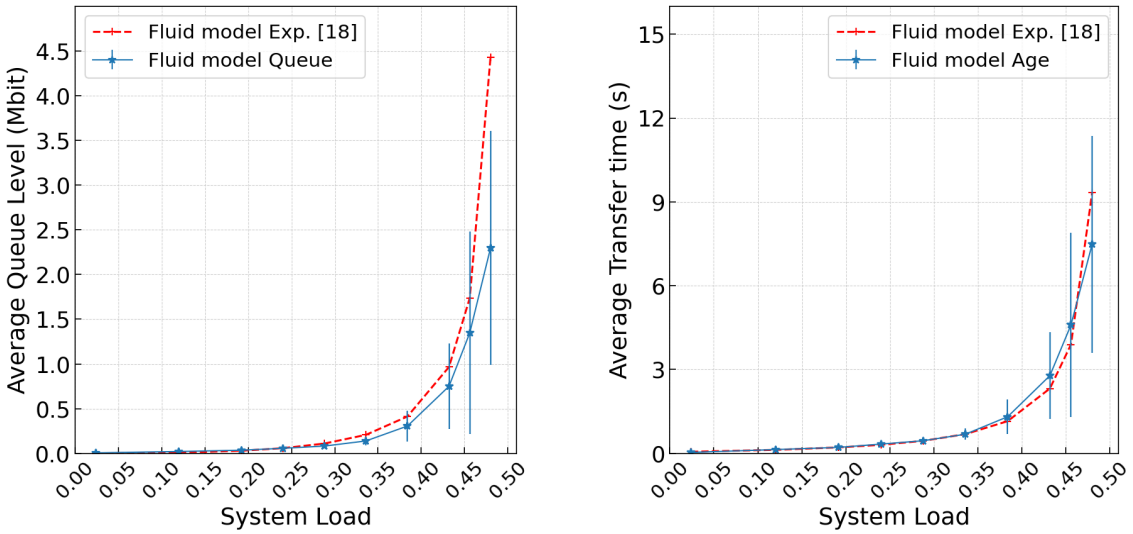
represent each network packet. In [133], the author proposed a fluid model for representing the behaviour of a bottleneck node in an ad hoc wireless network. Their model, which aims to estimate steady-state values, was implemented and validated with a discrete network simulator. The same scenario and models described in the paper were implemented and compared to our model, achieving similar results as shown in Figure 3.5 for an exponential distribution of the mean flow size.

In this scenario, mean flow size $f = 0.12Mbit$ and system capacity $C = 5Mbps$[2] are fixed, while the mean flow arrival increases to achieve a system load $\rho = (\lambda f)/C$ close to 0.5, which is the maximum stable load due to the fact that each arriving flow must be served twice before reaching the destination.

3.5 Implementation and Evaluation

In this section, the described models' implementations are used to evaluate and illustrate their ability to study data flow in mobile ad hoc distributed systems. Some stress-test traffic is injected with the dataflow models described in Section 3.3.5 so that the topologies reveal potential hotspots and the impact they have on hindering the overall system's performance. In order to evaluate the effect of the data flow, the injected traffic is gradually incremented through the network, configured as in Table 3.2 and runs in a regular *Intel i7-8550U* home computer with 16GB of DDR4 RAM. The evaluation then continues in subsections that expose different features, starting from congestion through latency and scalability. The evaluation was run with the three different[3] topologies illustrated in Figure 3.6.

[2] Approximate net capacity of a IEEE802.11b link
[3] With a fixed number of 18 nodes.

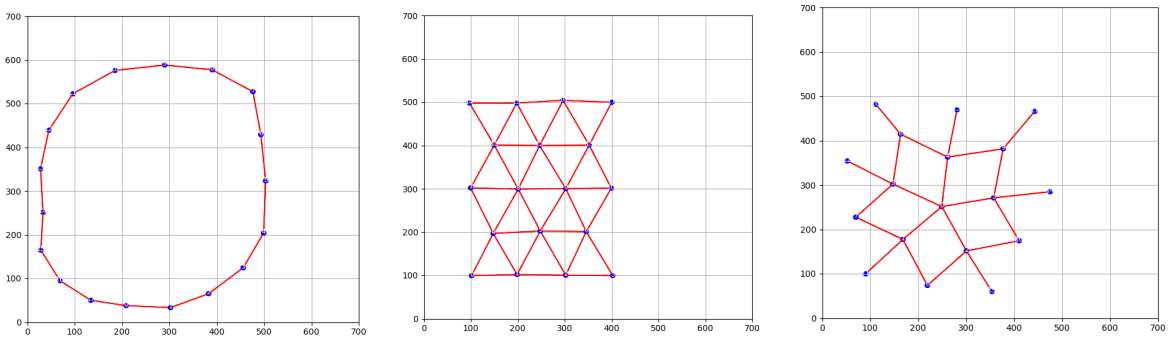


(a) Bottleneck Queue Stability Limit (b) Average Transfer Time

Figure 3.5: Model validation for average steady state queue level and transfer time for arbitrary particle during system load saturation.

Table 3.2: Network Configuration

Bandwidth:	54 Mbps
Time Horizon:	100 s
Processing Latency:	1 ms
Route update interval:	1 s



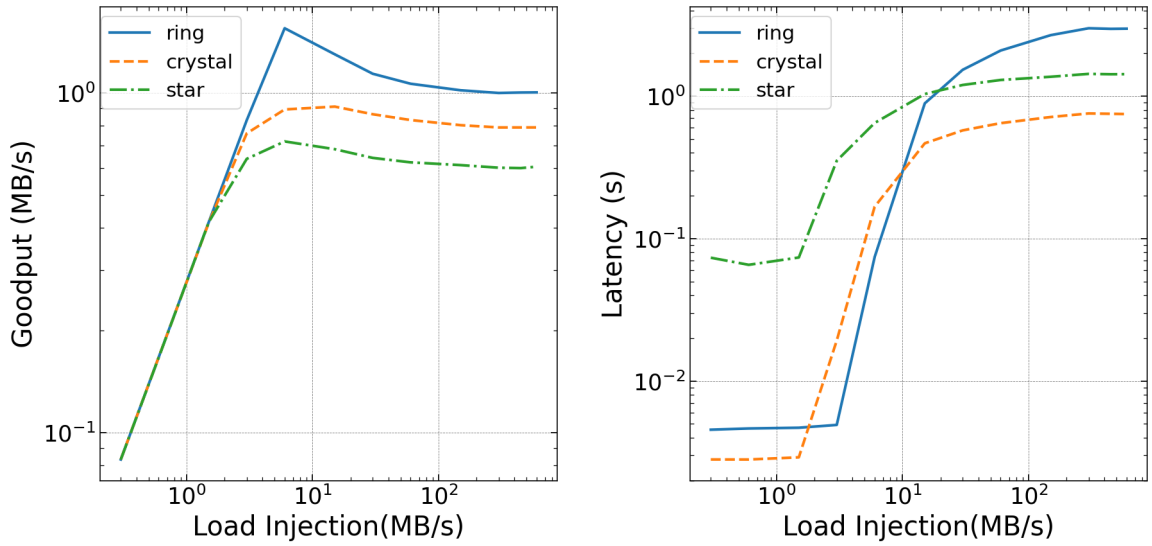
(a) Ring (b) Crystal (c) Star

Figure 3.6: Topology samples used for experimentation.

3.5.1 Data Throughput

Even though a mobile ad hoc topology can implement any type of distributed application, as long as the application requires that data transmission is associated with the execution of the required task or function, the introduction of network latency shall hinder the system's performance. In applications requiring data transfer, the data must be copied or moved before

the computation associated with such data can start. Some protocols optimize the data usage to avoid unnecessary traffic flow, but reducing unwanted latencies will allow the system to transfer more data in the same amount of time.



(a) Network saturation

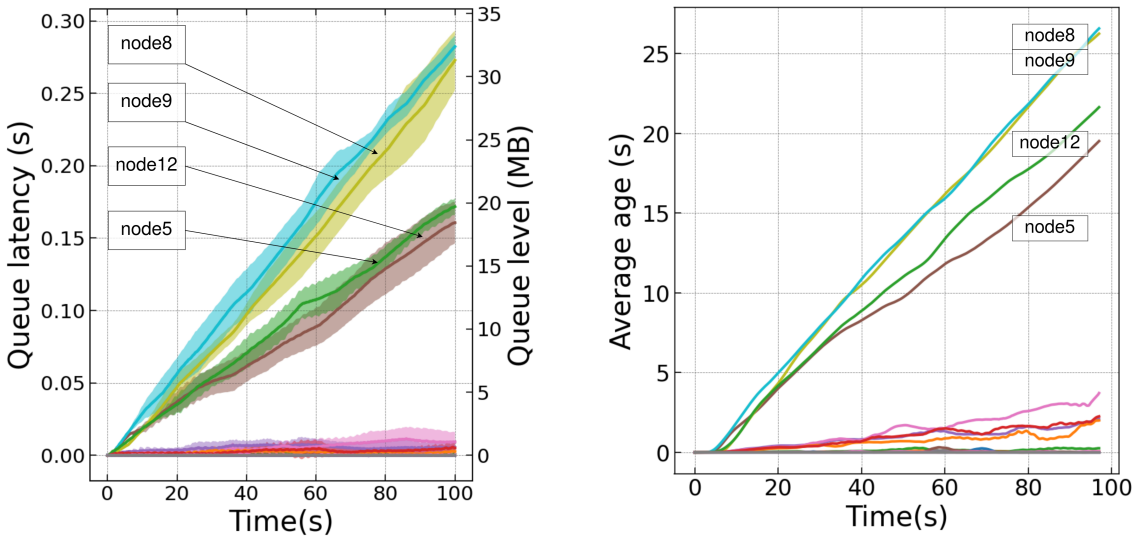
(b) Latency vs TP

Figure 3.7: Network saturation while injecting 3MB, *Round Robin* w/ variable frequency.

In Figure 3.7a, the goodput, which is the useful data delivered at the final destination, is represented as it grows together with the injected load until saturation occurs. Also, the topologies saturate at different traffic levels and with different steady-state levels of goodput. Such saturation can also be observed in the latency increase, as shown in Figure 3.7b. It is noticeable that regarding latencies, the topologies behave differently. This is especially evident in the ring topology, where the nodes do not have alternative routes, which results in higher latencies due to queue formation.

3.5.2 Congestion

As mentioned in 1.1, when running distributed applications in ad hoc networks, unbalances are likely to occur in some nodes due to the multi-hop traffic flow. Some end-to-end preferred routes will be established when running topologies, such as illustrated in Figure 3.6. Consequently, some nodes will have more data traversing them due to the data flow's characteristics. Such nodes are more prone to congestion, which might increase the end-to-end latency and, consequently, reduce the performance, as shown in Section 3.5.1. Since the first goal is to observe the formation of hotspots in the topologies, the queues in each node are modelled boundless, and no queue management algorithm is put in place. In Figure 3.8a each node was offloading 0.5 MB of data with the other nodes in the topology from Figure 3.6b in a round-robin scheduling and interval of 100ms.

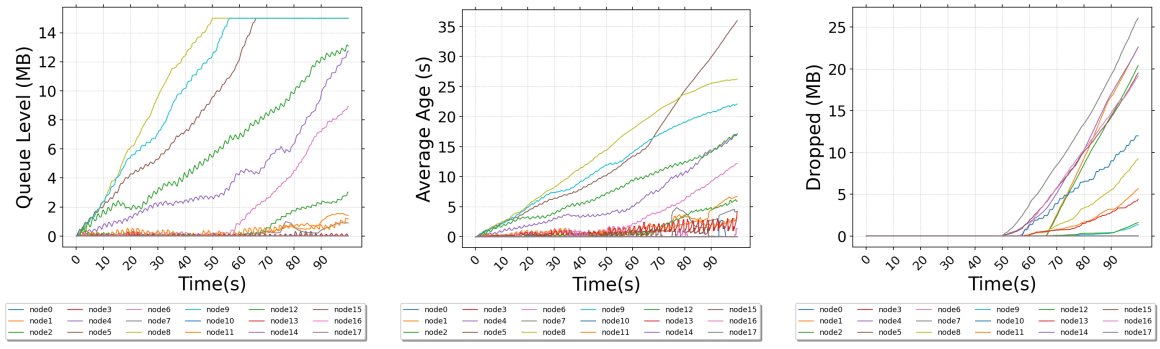


(a) Queue level and latency

(b) Average Age

Figure 3.8: Stress load injection of 0.5MB, *Round Robin* ($\lambda = 10$).

It is possible to observe that four nodes were the preferred routes of the data flow and allowed the formation of queues. The shadows in the plots represent the standard deviation of the collected data because since the mobility imposed on the nodes has a random component to their behaviour, each simulation result will differ slightly. In Figure 3.9 we configure the same scenario with Drop tail queue with maximum buffer of 15MB. As expected, the since the congestion point is limited, the queue formation propagate to upstream nodes, increasing also as a consequence the average age of the messages, as shown in Figure 3.9b. In Figure 3.9c it is possible to see that even though only three nodes are congested with full buffer, all the other neighbouring nodes are affected and start to drop messages, which could be mitigated with a congestion control algorithm informing them to reduce their sending rate.



(a) Queue levels

(b) Average Age

(c) Dropped Messages

Figure 3.9: Stress load injection of 0.5MB, *Round Robin* ($\lambda = 10$) and Droptail Queue with 15MB.

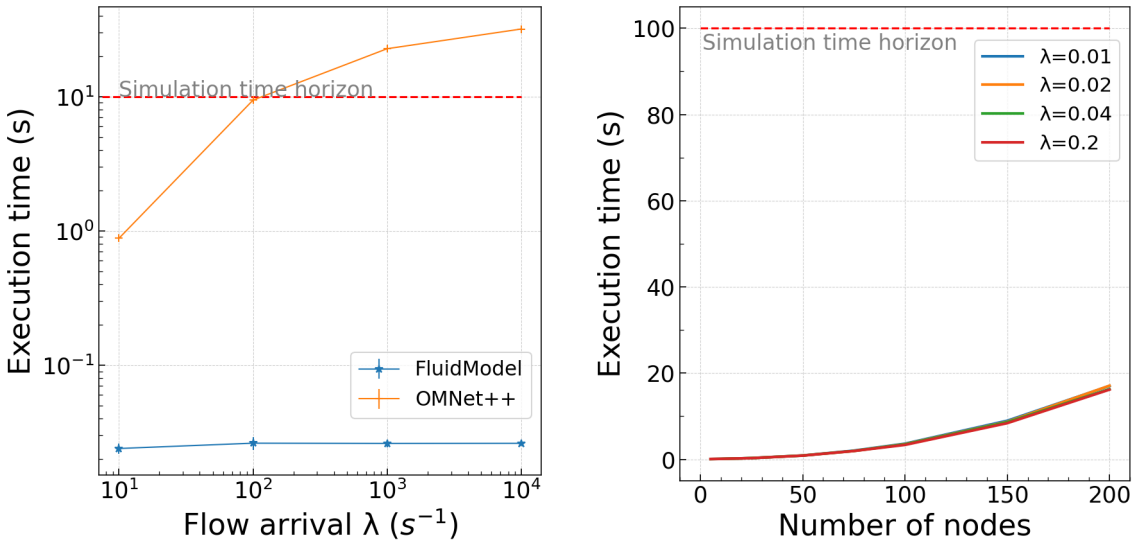
3.5.3 Latency

Communication latency is mainly characterized as described in Section 3.3.4 and Section 3.2.3. When abstracting unbounded queues in each node, due to the formation of transient or permanent bottlenecks, the network packets will undergo an extra latency besides the one created by the bounded bandwidth available for communication. When waiting in a queue to be served by the communication system, packets undergo an unexpected latency that can be estimated by Little's Law [90]. Figure 3.8a also illustrates the expected waiting time in each node when using Little's law. Even though this estimation gives an insight into additional expected latency, it is incomplete since it considers only the system's current state. The model described in Section 3.2.3, however, since it represents an average latency based on the history of the propagated end-to-end latency, can estimate the latter without the need to measure each message individually.

Figure 3.8b shows the average total latency expected for the messages measured in each node. It is possible to observe the higher magnitude compared with Figure 3.8a since it does not represent the expected added latency by each node's queue, but the average *age* of each message that has traversed the node.

3.5.4 Model Scalability Evaluation

One of the main advantages of fluid models compared to discrete models is scalability. The results in Figure 3.10a, which corresponds to a topology with five nodes, with three of them injecting flows of 1kB with an increasing arrival rate, shows that with OMNet++ the simulation runtime surpasses the time horizon, even with slow flow arrival. The runtime with the fluid model is barely influenced by the arrival rate, i.e. the number of network packets.



(a) OMNet++ with 18 nodes and variable injection

(b) Model with variable node number and injection

Figure 3.10: Scalability evaluation with time horizon of 100s.

Albeit not majorly impacted by the frequency of injected flows, the fluid model has runtime more impacted by the number of nodes in the topology. Figure 3.10b illustrates, for different arrival rates, a randomly deployed topology with increasing node count. Even though the runtime increases, it remains lower than the simulation time horizon.

3.5.5 Mobility Models

Since this thesis proposes observing distributed systems deployed in dynamic networks with time-varying topologies, the introduction of mobility models is required. With them, the nodes' positions periodically change, and the states of other models are updated accordingly by recalculating distances between nodes. The following mobility models are implemented for the evaluation in Section 3.5. A comprehensive list of mobility models that are useful to ad hoc computing, in particular when dealing with FANETs, can be found in [114], and since the implementation of this fluid model is modular, it is straight forward to implement new models.

Attraction Model

The attraction model was created to emulate the situation where a node is ordered to keep the position on a specific coordinate, but it is unable to execute the command exactly due to external perturbations, such as lateral wind. Consequently, the node fails to keep its exact position $\mathcal{P}(t) \in \mathbb{R}^3$ by an error that can be parametrized. To model disturbances, the attraction coordinates $\mathcal{A}(t) \in \mathbb{R}^3$ are a set of type $[x, y, z]$. Each Cartesian coordinate is periodically modified by a random, bounded variation $\epsilon(t) \in \mathbb{R}^3$ at a configurable period. At each time step, the position is changed by a $\delta_p(t) = \delta t / V_p(t)(\mathcal{A}_p(t) - \mathcal{P}_p(t))$ based on a pre-defined velocity $\mathcal{V} \in \mathbb{R}^+$, with $\delta_p(t) \in \mathbb{R}^3$ being a set $[x, y, z]$. Hence, the position is calculated by Equation 3.13 and, as a proportional controller, the closer the node gets to the attraction point, the slower it moves.

$$\mathcal{P}(t + \delta t) = [x(t) + \delta_{p_x}(t), y(t) + \delta_{p_y}(t), z(t) + \delta_{p_z}(t)] \tag{3.13}$$

Random Walk Model

The Random Walk model is more suitable when the goal is to have a mobility model that allows the nodes to move freely in a pre-defined area. It mimics applications where nodes do not have pre-defined mobility paths or when they have, but the scheduling of nodes in such paths is stochastic. In such a model, a limiting area $\mathcal{L}_a \in \mathbb{R}^3$ is pre-configured, and an initial position $\mathcal{P}(0) = [x_0, y_0, z_0]$ is defined in runtime. A random velocity $\mathcal{V}_0 \in \mathbb{R}$ is also defined in runtime, and it characterizes the initial direction given to each node. At each simulation time-step, the position is updated by a $\delta_p(t)$ (distance equivalent to the velocity and the time-step) and calculated as shown in Equation 3.13. When there is a collision of a node with a wall, the velocity has its signal inverted in the axis of the wall hit. As a consequence, the nodes keep bouncing inside the arena, as shown in Figure 3.11, but for simplicity, the collisions between nodes are ignored.

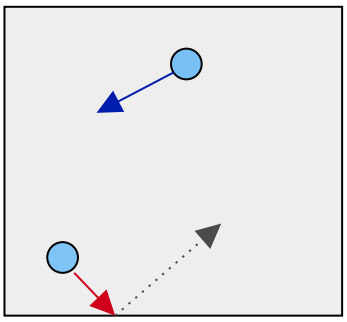


Figure 3.11: The Random Walk Mobility Model.

Motion Control Model

In addition to the possibility of using mobility models as defined by the authors in [45], a motion control model was also implemented for the mobility of the nodes. With this model, they can be controlled by the influence of different control laws, from which the outputs are velocity vectors. Therefore, each node's j position $p_j \in \mathbb{R}^M$ state is updated by an increment p'_j.

$$\dot{p_j} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) \\ sin(\theta) \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w \tag{3.14}$$

With $v \in \mathbb{R}$ being the velocity measured in m/s as the resultant of an applied force and $w \in \mathbb{R}$ the angular velocity measured in rad/s.

Follow Path Model

As a way to evaluate this thesis's contributions while the swarm performs a mission, a *Proportional Integral Derivative (PID)* controller was added to guide a node through a list of pre-defined waypoints, as shown in Figure 3.12. The output of the PID controller defines the magnitude and direction of the control vector, with gains defined so that the heading of a node changes smoothly towards the direction of the next waypoint.

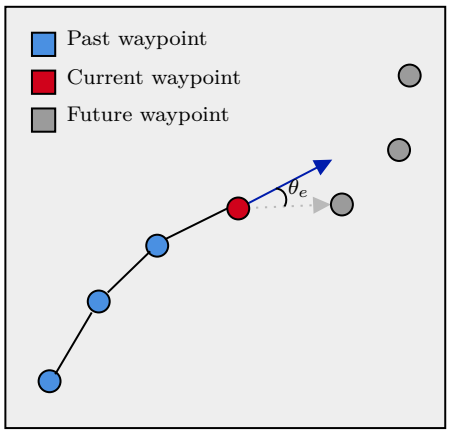


Figure 3.12: The Follow Path Model Calculates the Current Direction Error Towards the Next Waypoint.

Considering a node at position $(x(t), y(t))$ with next destination the way point with coordinates (x_w, y_w), the direction θ_d that it should be aiming is:

$$\theta_d(t) = atan\left(\frac{y_w - y(t)}{x_w - x(t)}\right) \tag{3.15}$$

The error in direction is therefore:

$$\theta_e(t) = \theta(t) - \theta_d(t) \tag{3.16}$$

The PID controller is finally defined by Equation 3.17.

$$u_{pid}(t) = K_p\theta_e(t) + K_i\int_0^t \theta_e(\tau)d\tau + K_d\frac{d\theta_e(t)}{dt} \tag{3.17}$$

By assigning one node to follow a path and act as a swarm leader, the others can equalize the heading and velocity by exchanging the state with their k-nearest neighbourhood. By using the consensus model described in Equation 2.10, eventually, they will follow the same path as the leader node.

3.6 Swarm Control Laws

As mentioned in Section 3.5.5, control laws can be modelled to control the mobility of the nodes. The final output of these control laws is a set of vectors that, when applied to the node's controller, induce a movement in the vector's direction with velocity proportional no the vector's magnitude. Due to the limitations in the self-propelled particle propulsion, this vector must be limited to a certain magnitude V_{max}, which is limited by the physical ability of the modelled robot.

Assumption 3. *The mobile robots current in position (x, y) and moving with direction $\theta(t)$ and velocity $V(t)$, assuming the do not exceed the maximum velocity V_{max}, can freely change direction towards the new direction commanded by the controller.*

Therefore, considering different controllers produce different output vectors as shown in Figure 3.13, the robot's controller will receive as input $\vec{U_r'}$, which is derived from $\vec{U_r}$ by capping the velocity with V_{max}.

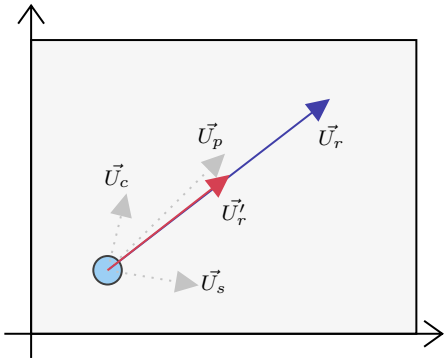


Figure 3.13: Combining the Vector Forces

3.6.1 Connectivity Maintenance Control

The connectivity maintenance control law has as its primary goal to maintain the connectivity, as described in Section 2.3.1, above a predefined threshold ϵ. The current connectivity, λ, can be calculated at any time as the second eigenvalue of the Laplacian Matrix from Equation 2.11. The control law, therefore, aims to compensate for the error $\mathbf{E} = \lambda - \epsilon$. To keep the value from going below ϵ, when $\mathbf{E}$ approaches zero, the energy function $V(\lambda)$, initially proposed in [136], tends to infinity when the error tends to zero, and converges to a constant when the connectivity is high, as shown in Figure 3.14.

$$V(\lambda) = \begin{cases} coth(\mathbf{E}), & \text{if } \lambda > \epsilon \\ 0, & otherwise \end{cases} \tag{3.18}$$

Therefore, the rationale behind the controller is to calculate the gradient of the energy function 3.18 and create a force in the opposite direction, as shown in Equation 3.19.

$$u_i^c = \frac{\partial V(\lambda)}{\partial d_i} = \frac{\partial V(\lambda)}{\partial \lambda}\frac{\partial \lambda}{\partial d_i} \tag{3.19}$$

With,

$$\begin{aligned} \frac{\partial V(\lambda)}{\partial \lambda} &= -csch^2(\lambda - \epsilon) \\ &= -\frac{4}{(e^{(\lambda-\epsilon)} - e^{-(\lambda-\epsilon)})^2} \end{aligned} \tag{3.20}$$

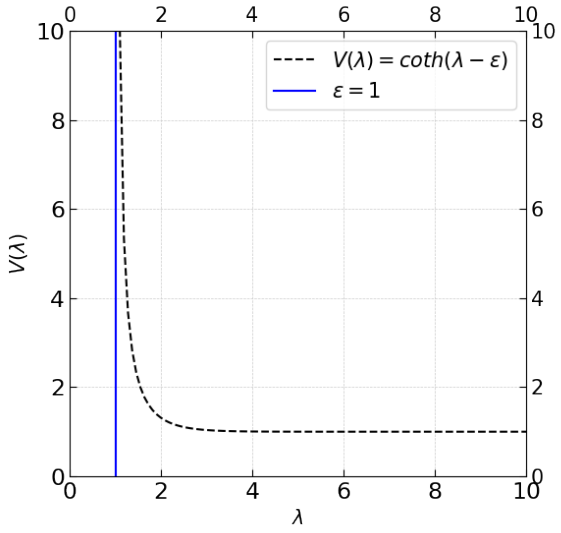


Figure 3.14: Energy Function used by the Connectivity Controller

3.6.2 Collision Avoidance

Collision avoidance is an important aspect when modelling more realist swarms of CPS. It is crucial for the availability of the systems that the nodes are free to move in space with a low risk of collision. The rationale is that when nodes are in collision route, a strong opposing force is created that forces the nodes to avoid the collision while they are still able to move.

Artificial Potential Fields

Artificial potential fields have been used in swarm control for two main applications, path planning and collision avoidance. The idea comes from the electromagnetic fields that can create repulsion or attraction forces. Artificial potential fields can be used, for instance, to create repulsive forces in known obstacles in the region where the swarm is deployed and to define inter-node forces. In this thesis, we focus on collision avoidance between the nodes by using a model proposed initially to model molecular dynamics.

Lennard-Jones Model

Originally proposed as a molecular dynamics model, the Lennard-Jones model has been used in many scientific applications to model realistic repulsion phenomena with a low computational budget. The potential field between nodes n_j and n_k, located at (x_j, y_j) and (x_k, y_k) respectively, is defined by Equation 3.21.

$$V(|k-j|) = \epsilon \left(\left(\frac{\sigma}{|k-j|} \right)^a - 2 \left(\frac{\sigma}{|k-j|} \right)^b \right) \tag{3.21}$$

Where σ is the goal distance where the force will grow exponentially, ϵ defines the intensity of the repulsive force, and a and b are constants that, on the original model, depend on the

type atom studied. They ultimately define the shape of the curve illustrated in Figure 3.15.

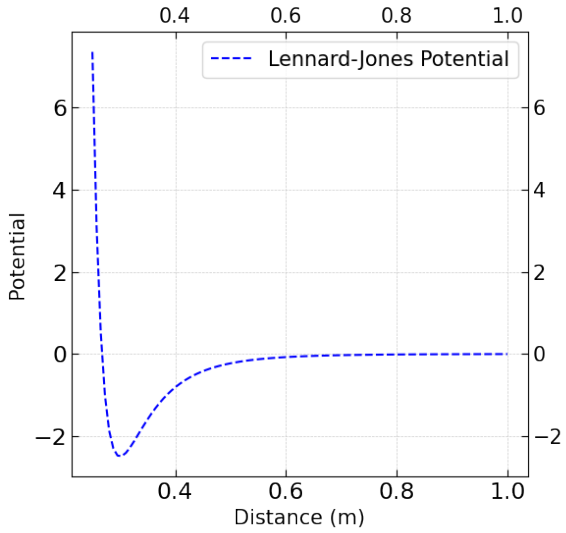


Figure 3.15: Lennard-Jones Potential Field. $a = 12$ $b = 6$ $\epsilon = 2.5$ $\sigma = 0.3$

From Equation 3.21 it is possible to derive the actual vector force that can be used by a controller, as shown in Equation 3.22.

$$u_i^p = -\epsilon \left[\left(\frac{a\sigma^a}{d^{a+1}} \right) - 2 \left(\frac{b\sigma^b}{d^{b+1}} \right) \right] \tag{3.22}$$

3.6.3 Consensus Controller

The consensus controller aims to match the direction and velocity of all nodes in the swarm. Using the principle explained in Chapter 2.2.4, a convergence consensus is used with the goal that their state converge. The state S is composed of the node's current velocity N_v measured in m/s, and their direction N_θ measured in radians. So, in a swarm of CPS, each node i broadcasts its current state to its 1-hop neighbourhood. Hence, each node builds a neighbourhood table N_i, with which they continuously exchange their state. At each unsynchronized information exchange round, the nodes use Equations 3.23 and 3.24 to update their local state.

$$\dot{\theta}_i(t) = k \sum_{j \in N_i} aij(t)(\theta_j(t) - \theta_i(t)) \tag{3.23}$$

$$\dot{v}_i(t) = k \sum_{j \in N_i} aij(t)(v_j(t) - v_i(t)) \tag{3.24}$$

Figure 3.16 illustrates one example topology where we can apply the consensus controller.

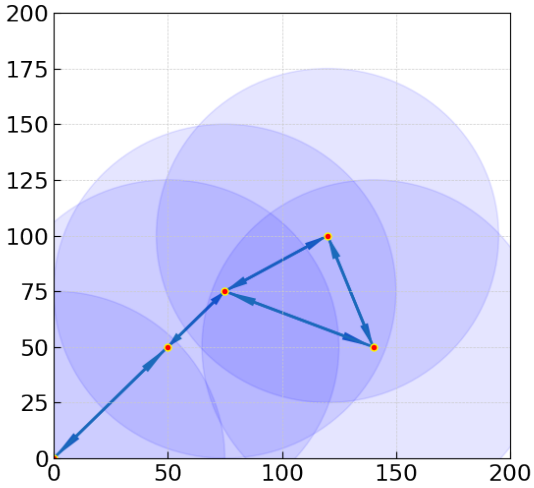
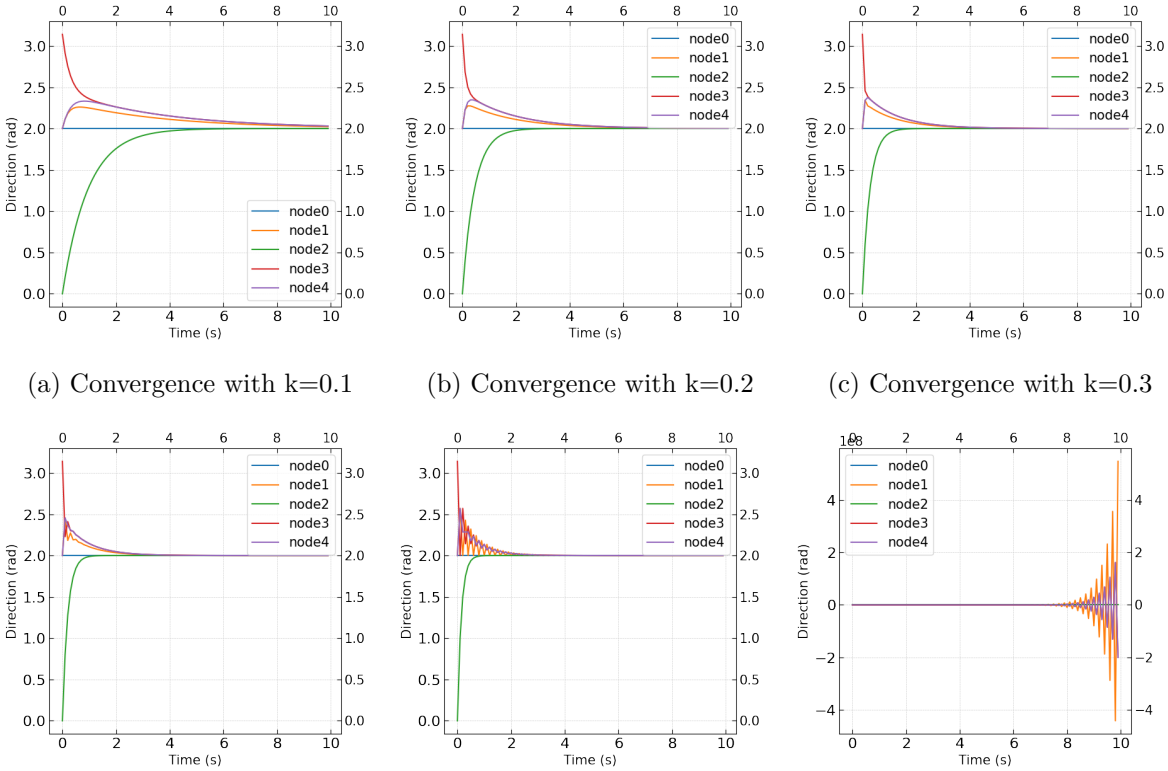


Figure 3.16: Example Graph with Radio Range Representation.

This topology creates the following adjacency matrix.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{3.25}$$

We simulated that nodes exchanged information about their state every 100ms and kept n_0 constant as a leader. The results of the evolution of their direction are shown in Figure 3.17. As expected, increasing the constant k, increases how much the state is updated at each communication round. However, increasing it too much leads to instability, making the consensus impossible.



(a) Convergence with k=0.1 (b) Convergence with k=0.2 (c) Convergence with k=0.3

(d) Convergence with k=0.4 (e) Convergence with k=0.5 (f) Convergence with k=0.6

Figure 3.17: Effect of the k Gain on the Convergence Controller. Update period t=100ms

3.6.4 Combining the Controllers

As mentioned at the beginning of Section 3.6, each controller works independently, but their output can be combined by performing a weighted sum as shown in Equation 3.26.

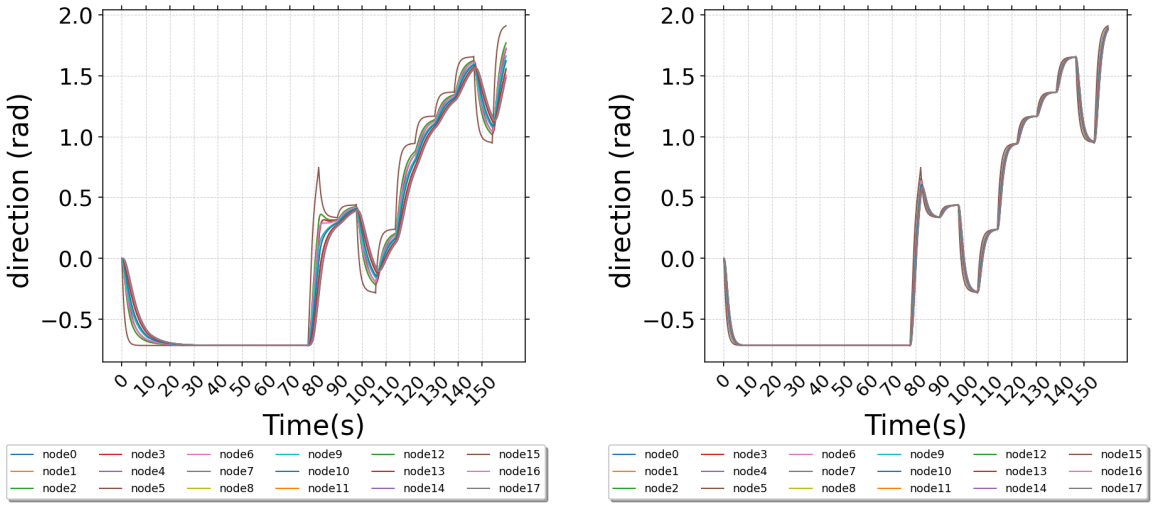
$$\vec{U} = \alpha\vec{U_f} + \beta\vec{U_s} + \gamma\vec{U_c} + \eta\vec{U_p} \tag{3.26}$$

Each controller has one configurable weight associated $\alpha \in \mathbb{R}^+$, $\beta \in \mathbb{R}^+$, $\gamma \in \mathbb{R}^+$ and $\eta \in \mathbb{R}^+$, with $\alpha + \beta + \gamma + \eta = 1$.

- $\vec{U_f}$ - Follow Path, with gain α
- $\vec{U_s}$ - Swarm Consensus, with gain β
- $\vec{U_c}$ - Connectivity Maintenance, with gain γ
- $\vec{U_p}$ - Potential Fields, with gain η

Figure 3.18 illustrates the direction in radian of each node when one node's mission is to follow a predefined path, and the other nodes match speed and direction by using the convergence consensus controller. In Figure 3.18a, it is possible to see that having a low gain on the controller induces some error in the direction that eventually dissipates before the next

way-point. Increasing the gain induces a more *tight* control but could interfere with the other controllers.



(a) Convergence Consensus Configured with low Gain of 1.

(b) Convergence Consensus Configured with high Gain of 5.

Figure 3.18: Direction Representation of Node 6 Using the Follow Path Model.

3.7 Case Studies

It is reasonable to assume that it is possible to use a swarm's mobility capacity to modify the traffic flow behaviour and hence also to use it to balance the traffic in the system. Thus, removing bottlenecks and, consequently, enhancing the overall application performance. To evaluate this possibility, a simple scenario was implemented as a concept.

3.7.1 Mobility to Enhance the Traffic Flow

The concept is based on a control system since it controls the positioning of some nodes in the topology based on real-time measurements. The proposed system could even be implemented in a distributed architecture. The rationale is that each node measures its own queuing levels and latency estimation in real-time. Nodes with low queueing levels and low average age levels are nodes with low levels of traversing traffic. On the other hand, a node with high levels of traversing traffic can see a trend of growing queuing and request a change in topology. A simple derivative can be applied to the queue level to detect an upwards trend and compare it with its neighbours. So, if we consider the current queue level as a proxy measurement for the flow $\Psi \in \mathbb{R}$, and assume that it is always differentiable, the queue derivative $\dot{Q}_n$ will indicate when inflow is too fast, and an action should be implemented.

Assumption 4. *The queue levels are always continuous and differentiable.*

Therefore, each node n needs to constantly calculate the derivative $\dot{Q}_n$ of their queuing levels and broadcast to their neighbours.

Protocol Description

As described in Algorithm 3, when their own derivative value is higher than the average of its neighbours by a configurable factor $\psi \in \mathbb{R}^+$, they trigger a topology-wise broadcast message $M_r = \{id, p_n(t_t)\}$. The message contains a demand for support identified by a unique id and their last know position p_n at the time of trigger t_t. All nodes must mark themselves as available following Algorithm 1. If their queue level and average age are below a configurable threshold and they are not currently supporting another node, they mark themselves as available.

Algorithm 1 Define if Available to Support

```
a ← false                                              ▷ Available
k_q                                                    ▷ Queue level threshold
k_α                                                    ▷ Average Age threshold
s                                                      ▷ Currently supporting
while true do
    if (s == false) then
        if (Q_n < k_q) AND (α_n < k_α) then
            a ← true
        else if (Q_n >= k_q) AND (α_n >= k_α) then
            a ← false
        end if
    end if
end while
```

Algorithm 2 Deliver Support Demand

```
t_b                                                    ▷ Backoff time
s                                                      ▷ Currently supporting
DELIVER(M_d^k)                                         ▷ Delivers a request to support
if (a_n == true) then
    BROADCAST(M_a = {n_id, d_nk(t_t)})
    s_n ← true
    a_n ← false
    while t_b do                                       ▷ While backoff timer have not expired
        DELIVER(M_a^j)                                 ▷ Deliver all possible M_a from other nodes
        RESET(t_b)                                     ▷ Reset backoff time
        if (d_jk(t_t) < d_nk(t_t)) then                ▷ If other node is closer
            s_n ← false                                ▷ Disable support
        end if
    end while
    SEND_SUPPORT()
end if
```

When a node delivers the broadcast and has marked itself as available, it follows Algorithm 2 and broadcasts to the topology an available message $M_a = \{n_{id}, d_{jk}(t_t)\}$, attaching the distance between itself and the last known location of the node requesting support. It then

waits for a configurable backoff time t_b for another broadcast with a shorter distance. Until it receives support, the node keeps broadcasting its current position so that the supporter can follow it. When the supporter reaches the supported node by a distance Δ that will not create too much contention, it follows the node, and as a consequence of the routing protocol, they share the traffic. When the supported node receives the support, it disables the request for support and stops broadcasting its location.

Algorithm 3 Detect and Broadcast Demand for Support

```
a ← false                                              ▷ Available
k_q                                                    ▷ Queue level threshold
k_n                                                    ▷ Neighbourhood threshold
A_n                                                    ▷ Neighbourhood average
s                                                      ▷ Currently being supported
while true do
    A_n ← CALC_AVERAGE()
    if (Q̇ > k_q) AND (Q̇ > A_n * k_n) then
        a ← false
        s ← true
        BROADCAST_REQUEST(M_d^k)
    end if
end while
```

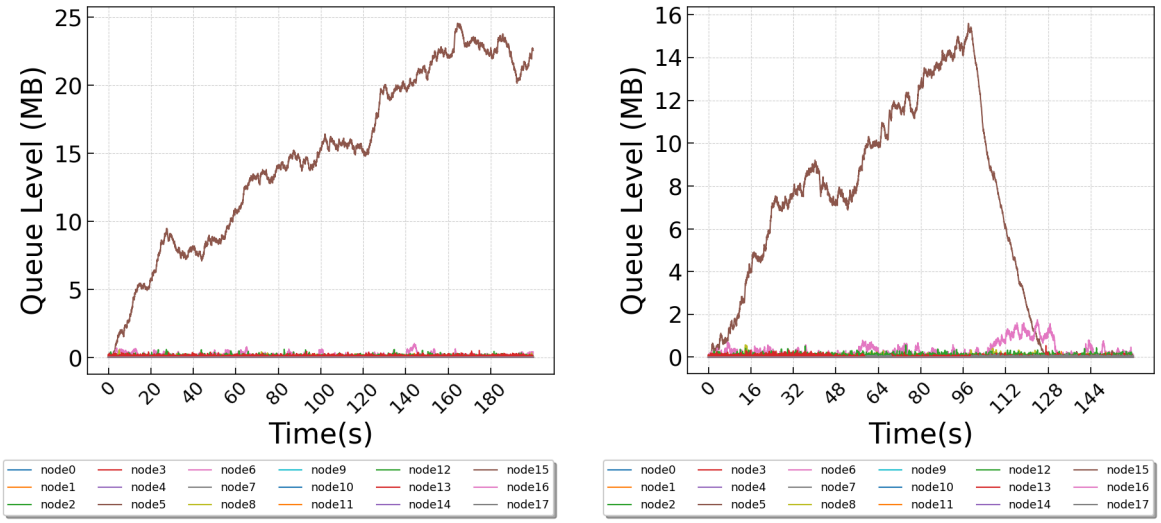
Results

To evaluate the feasibility of this study case, we implemented the algorithms to run in the fluid model simulator. The simulations were run with the parameters listed in Table 3.3.

Table 3.3: Simulation Parameters

Bandwidth:	54 Mbps
Time Horizon:	160 s
Queue Level threshold (k_q):	5 MB
Neighbourhood threshold (k_q) :	2
Max. Velocity:	3.5 m/s
Mobility:	Static
Swarm Consensus gain $\vec{U_s}$:	0
Connectivity Maintenance gain $\vec{U_c}$:	0
Potential Fields gain $\vec{U_p}$:	0
Data Model Source:	All nodes
Data Model Target:	Node 9
Data Model Period:	0.11 s
Data Model Payload:	0.06 MB
Queue Limit:	30 MB
Queue Model:	Drop Tail

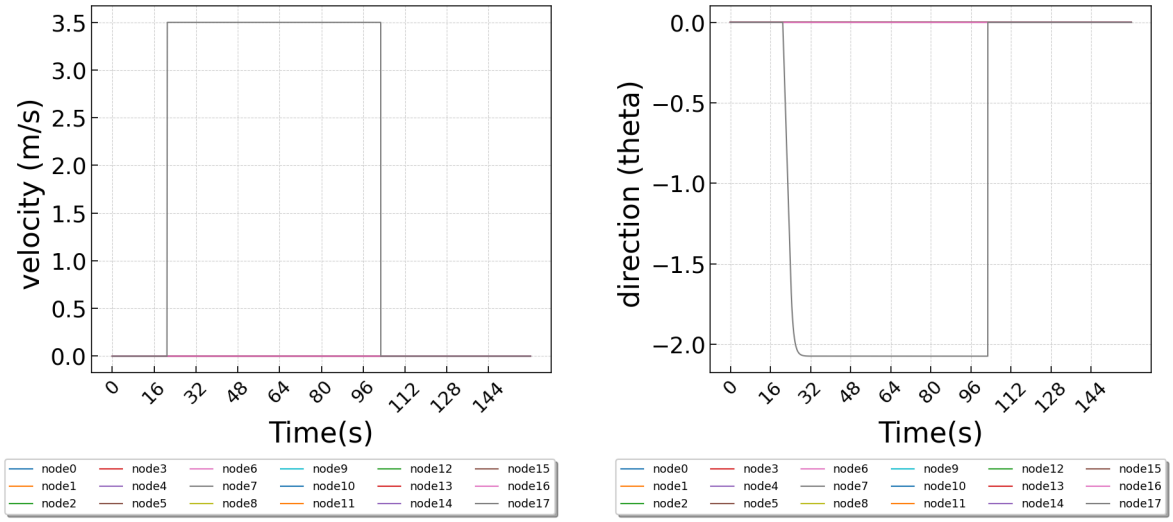
Figure 3.19 shows that changing some nodes' position in the topology impacts traffic flow and allows mitigation of bottleneck formation. That allows a better network flow with an increase of 2% in total data delivered by the system, which could be enhanced if the topology was defined as so from the beginning.



(a) Queue Levels Without Support

(b) Queue Levels With Support

Figure 3.19: Evolution with topology adaptation after XXs



(a) Velocity With Support

(b) Direction With Support

Figure 3.20: Evolution with topology adaptation after XXs

3.7.2 Modelling Quorum-Based Consensus

To simulate traffic created by a quorum-based consensus algorithm such as Paxos, an event-based model was defined to run together with the fluid model. The model allows the definition of a leader node serving as a proposer and a set of nodes that serve as acceptors and learners. It works by scheduling transmissions of proposals based on a start time and periodicity. The

subsequent messages, such as the promise, accept and accepted messages, are scheduled as shown in Figure 3.21, based on the current estimated end-to-end delays.

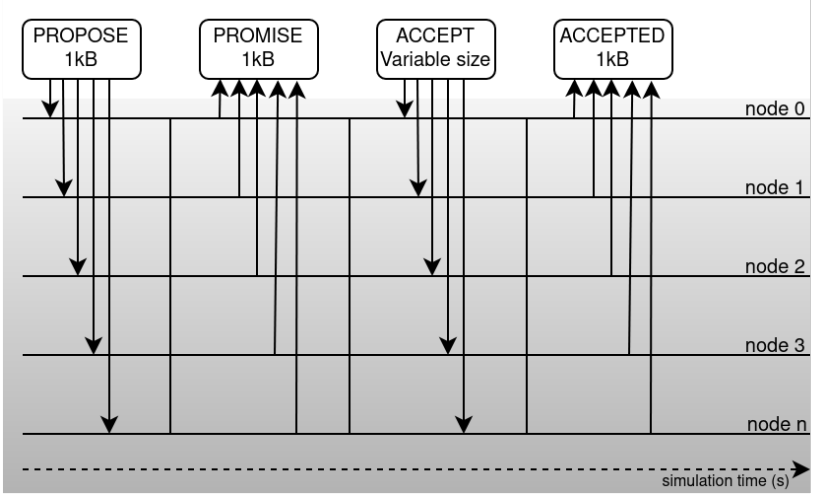


Figure 3.21: Paxos Communication Sequence

The size of each type of message can be previously configured, and the messages are injected into the fluid model affecting the data flow and, consequently, the queue levels.

3.8 Related Work and Limitations

In [16] and [133], the authors proposed a fluid flow model for representing a bottleneck being served by multiple sources, while serving one node. Later, in [134], the authors expanded the work by also modelling resource-sharing techniques for overall performance enhancement. In addition to modelling such bottlenecks, our work went further with the ability to represent flows with different routes and considering nodes' mobility, hence being suitable for dynamic networks. Furthermore, our model allows the study of transient queue and latency dynamics during the whole execution, not only steady-state. The work presented in [154] also introduced a fluid model for ad hoc networks, but with focus on energy consumption. In the domain of static networks, other works created fluid models based on differential equations, such as in [100] and even the creation of a framework in [18]. Our model is also inspired by the fluid model described in [34], and their implementation was used as a scaffold for our implementation. The main limitation, when compared to discrete simulators, is when there is a need to investigate detailed network behaviour and implement different wireless stacks. Our model does not implement detailed contention algorithms, so its overhead must be taken into consideration by using an approximated usable bandwidth value. The current model is insensible to flow size i.e. long lived or short lived bursty traffic, so different average flow sizes yield similar results.

3.9 Conclusion and Future Work

In this chapter, a lightweight analytical model was introduced, implemented, validated and tested. Its applicability as a tool for faster than real-time simulations of different deployment topologies for swarms of mobile distributed systems was demonstrated. The model, even running on a personal computer, can simulate hundreds of nodes in less time than the simulation time horizon, demonstrating its scalability. We extended the model by adding mobility control based in swarm consensus and control laws, expanding the possibility of research in swarm intelligence. We also implemented some case studies exploring the possibility to model distributed algorithms to introduce traffic flow, and explored the possibility of adapting the position of the nodes to mitigate congestions. We also plan to update the model so that it is also sensitive to long and short lived traffic flows; to implement message dropping after a maximum number of retries by modelling a probability function; and to design a gossip routing protocol. For applications of the model, future work will allow research on self-stabilizing topology control mechanisms, optimized node positioning and replica placement schemes for distributed systems. Due to its modularity, the work can also be expanded to study QoS and routing protocols. Other works have already modelled MAC protocols that could be incorporated to our simulator so that we take also in consideration the overhead introduced by them at each hop.

Chapter 4

Designing a Realistic Mobile Ad-hoc Computing Emulator

Contents

4.1 Introduction

As mentioned in Chapter 1, there has been a recent interest growth in mobile distributed computing, especially in the areas of *Mobile Cloud Computing (MCC)* [125], MEC [39] and *Mobile Ad-Hoc Computing (MAC)* [175]. This growth was mainly driven by the adoption of IoT, Edge and Fog applications and the market demand for low latency applications in the edge such as *Virtual Reality (VR)*/*Augmented Reality (AR)*, gaming and *Artificial Intelligence (AI)* applications for autonomous systems [109]. Many simulators and emulators were created to support this type of research, [98]. However, there is a lack of features in existing emulation tools available for mobile Edge/Fog computing, with some essential features missing. Among those missing features, a major one is the possibility of having a full-scale emulated deployment where prototypes coexist with off-the-shelf applications or more options for mobility control.

Additionally, containers and virtual machines running together with *Software Defined Networks (SDNs)* emulators have enabled system designers to run several nodes of distributed computers by emulating the topology of a wired network. However, when the application focuses on nodes running on a MANETs or a FANETs, some difficulties arise with the mobility representation and the behaviour simulation of the nodes' unstable connectivity. We, therefore, propose an emulation framework that enables the deployment of mobile distributed applications in a virtual environment, so that system designers and developers can easily modify the scenarios and topologies composed by mobile wireless nodes. As mentioned in Section 2.4, an emulation tool was also ideal for enabling observations of how existing state-of-the-art protocol implementations and off-the-shelf software would behave in the scenarios we envisioned concerting distributed systems running in dynamic networks.

Even though simulation platforms are very useful in early-stage development, emulation introduces the possibility of running released software or software in late development stage so that a physical device can easily substitute an emulated node without needing to change the software. Most of the simulators, cf. Section 4.2, require platform-specific development with custom libraries and have as goal creating detailed models of each layer of a network stack. Furthermore, the granularity supplied by the simulators often leads to high resource consumption from the host computer, which creates scalability obstacles. Also, since in this thesis the focus is not on studying the network stack itself but instead distributed applications, there is no need to simulate complex network behaviour on packet level.

Finally, the primary motivation behind developing this emulation tool was the need for a wireless emulation software which allows designing and testing mobile distributed algorithms and software, such as distributed coordination services, distributed data stores, task scheduling and offloading. When designing a mobile distributed system connected via wireless interfaces, new requirements arise that cannot be fulfilled by existing tools. Section 4.2 illustrates available solutions that also aim in IoT, Edge and Fog environments. The main features that could not be altogether fulfilled by any existing tool were: support of mobile wireless nodes through emulation; flexibility in the creation of new topologies such as positioning of the nodes and mobility models; scalability that allows both a higher number of nodes and high network traffic; possibility of running applications directly in the host, in virtual machines or containers and the ability to emulate constrained nodes; capacity to be integrated with external software for mobility control.

As for the time this thesis was published, it is possible to emulate deployments with

nodes running x86 or ARM virtual machines, Docker containers, IoT nodes running RiotOS or Raspbian, and applications running directly on the host operating system.

4.1.1 Fog, Edge and IoT Emulators and Simulators

Fog, Edge and IoT simulators and emulators usually have a network simulator or emulator as the main component but add some useful functionalities when researching in this field. Such functionalities usually relate to node management, resource allocation or data placement. They can also add valuable metrics, such as energy and cost calculations, that resource allocation, load balancing and scheduling algorithms can potentially exploit.

FogNetSim++ [124] is a toolkit aimed at testing Fog applications in mobile wireless networks that has the disadvantage of requiring the applications to be developed inside the network simulation framework. To work around this limitation, in [43] the author expanded the work of FogNetSim++ by connecting it to NodeRed[1], which allowed users to run realistic NodeRed applications in this simulated environment. Virtual Mesh[157, 158] allowed the creation of simulated nodes on a mesh network by simulating the network on a currently outdated version of a network simulator. Miniworld [144], on the other hand, uses a network emulator that can be used to create topologies of wireless nodes. It enables the execution of virtual machines in such emulated nodes, but it still lacks the flexibility of more complex mobility scenarios. In [64] iFogSim is proposed as a toolkit focusing on resource management. In contrast, in [107] the author extended the work to add a data placement extension to enable users to experiment with different policies. MockFog [65] proposes an emulation environment deployed in existing cloud infrastructure providers to allow the configuration of a heterogeneous environment. However, it is not suited for emulating mobile Ad Hoc nodes due to the limitations in the topology definition and lack of mobility control. It also cannot be used to emulate constrained devices. EmuFog [101], like this work, uses a network emulator to allow the deployment of several nodes in a controlled environment. The emulator used is MaxiNet, and the applications are emulated in a Docker container, as can be done in MACE but lack the option of adding mobility to the nodes. EmuEdge [179] is an emulator with similar goals of creating realistic experiments. It uses *Xen* to create virtual nodes and SDNs for creating and managing the topologies. It also allows detailed network tracing and replaying traffic generated by network simulators but lacks good wireless and mobility implementations. It also lacks the possibility of emulating constrained IoT nodes, focusing on Linux and Android VM nodes.

Another possibility of testing distributed prototypes in a realistic environment is to use test beds such as IoTLab [2], which allow the download of applications on physical embedded hardware and the observation of realistic radio behaviour. More details about these tools are discussed in Section 4.3. Table 4.1 shows what is, to the best of our knowledge, the current state of the art and illustrates the features it does not cover.

[1] https://nodered.org/about/

Table 4.1: Emulators Comparison

Emulator \ Functionality	Network Emulator	Mobilty Control	Distributed Mobile Applications					Energy Model	Resource Management & Schedulling	Open Source	Last Commit
			Linux Native Applications	ARM VMs	x86 VMs	x86 Containers	Embedded Applications				
FogNetSim++ [124]	OMNet++	✓						✓	✓	✓	2019
FogNetSim++ w/ NodeRed[43]	OMNet++	✓					NodeRed	✓	✓		
iFogSim [64, 107]	Built-in	✓						✓	✓	✓	2021
Miniworld [144]	CORE			Possibly	✓	✓				✓	2017
Virtual Mesh [157, 158]	OMNet++	✓	✓					✓			
MockFog [65]	Cloud Infra			Possibly	✓	✓			Since v.2	✓	2020
EmuFog [101]	MaxiNet					✓			✓	✓	2020
EmuEdge [179]	Linux veths	✓		Android	✓	✓				✓	2020
MACE	CORE	✓	✓	✓	✓	✓	RiotOS			✓	2023

4.1.2 Main Contribution

This chapter has been published as a conference paper[46], which had as its main contribution establishing a framework that enables researchers and engineers to work with all the aforementioned features in a simplified way. It not only enables the study of distributed mobile applications, but it also allows faster prototyping since it does not require additional software-specific skills for designing applications. The main functionalities are listed below:

- Simplified mobile wireless Ad hoc emulation with scenarios and topologies easily configured via GUI.
- Emulation of distributed systems composed of Virtual Machines, containers or applications running directly on the host OS.
- Integration with other tools that can provide mobility models or the possibility of connecting with other domain-specific simulators that can provide geo-location information for the nodes.
- An easy-to-use user interface that can be used for: creating new topologies, observing in real-time the current position of the nodes, and interactively changing the nodes' position.

A demonstration of a mobile system running an etcd [42] configuration service is presented as an experimental evaluation in Section 4.6, where it is compared to a prototype key-value store. The source code for the emulation topology and the prototype are also available. The source code of this framework is available at: *https://github.com/brunobcfum/ pymace*.

4.2 Background

As will be presented in Section 4.3, different emulators and simulators have been developed with Edge and Fog in mind, and the network component is one of the most important in such tools. This section presents some concepts and some of the most common of such tools.

4.2.1 Network Simulators

It is common to use network emulators or simulators to evaluate distributed algorithms. Network simulations are further divided into discrete and continuous, being the discrete network simulation the most common and popular type [150]. In this type of simulation, time can be sliced and counted in ticks that are set to happen with a predefined frequency. Otherwise, they can also be event-based, scheduling new events as they are triggered and updating clocks accordingly. In the former, the period can be set to be close to real-time or not, giving the user the ability to choose the simulation speed. In the latter, events happen immediately when possible, and time is counted based on the known period of such events. One advantage of this type of simulation is the possibility of running simulations much faster than wall time and reducing the computer's idle time as long as there are resources available. These types of simulators, such as OPNET, OMnet++ and NS-3, allow the creation of the whole network stack as base simulated blocks. The user can choose which components are needed and create

a stack according to their simulated scenarios, using different protocols and technologies previously developed or creating their own. The main drawback of such simulators is that new applications and protocols must be developed inside the framework and can only be directly ported to a host operating system with an inevitable rework. For example, an application built to be tested on OMNet++ can only be directly used in NS-3 with some refactoring.

The continuous network simulators are built when the main focus of the study is to analyze traffic flow that is represented as a mathematical model comprised of ordinary or partial differential equations [99]. In these types of simulators, usually, fluid mechanics models are adapted to the use of networks, and the main goal is to study how the traffic of packets flows in the network, not how each packet individually behaves. Hence, the models focus on the macroscopic characteristics of the flow. In Chapter 3 we propose one of such fluid models to be used in different scenarios than the one proposed in this chapter.

4.2.2 Network Emulators

Another class of software are network emulators, such as GNS3 [58] and CORE [31], which unlike the simulators mentioned before, aim to mimic the device's behaviour without modelling in detail each intrinsic aspect. On the other hand, the replacement of an emulated network interface is imperceptible by the applications. So, they allow the system designers to experience a more realistic approach for the tests since the emulators provide for the running application the perception that it is running on real hardware. In Linux environments, this is usually achieved by using virtual devices (TUN, TAP or VETH). They create a connection between kernel space and user space, allowing user space software access to lower levels of the network stack usually available only for the kernel [72]. Both NS-3 [110] and OMNet++ [66] have added the capability to emulate network devices by also using TAP interfaces and simulating the physical environment with discrete events scheduled by a real-time scheduler that tries to keep up with the computer wall time. Some problems arise when using discrete simulators with emulation capabilities such as OMNet++ and NS3 to emulate *close to real* deployment scenarios. Due to the difficulty of synchronizing the simulator scheduler with the computer wall time, when network traffic is high and there are too many events to be processed, many packets are dropped. This behaviour will be further discussed in Section 4.5.

4.3 State of the Art

This section presents a review of prior work relevant in the context of emulation for Edge/Fog environments, highlighting why they cannot fully meet the requirements for fast, easy-to-instrument mobile distributed application development.

4.3.1 FogNetSim++

FogNetSim++ [124] is a simulator built with OMNet++ that proposes an open-source toolkit for modelling new fog applications. This simulator fails, like many others, to allow simulated applications to be directly deployed without needing to adapt to an environment outside the OMNet++ simulator. In [43] the author expands the usability of FogNetSim++ by

introducing a communication channel[2] between the simulator and the application NodeRed[3]. The concept allows users to develop applications and test them in a simulated platform before deployment. The author's goal was to eliminate the intermediate step of developing the application twice, once for simulation and again for deployment. The proposed platform allows the creation of applications on a market-accepted platform for developing IoT applications and their integration into the simulation platform with minimal adjustments.

The concept idealized by the author is similar to one of the goals we had: code once, simulate and deploy. This work also aims to introduce an emulation platform that allows users to create their applications without having to produce an emulation platform-dependent application code. However, we aimed to go further by providing the option of running not only applications aimed at IoT devices but also possible *Virtual Machines (VMs)* and containers that can run in edge clouds or cloudlets.

4.3.2 iFogSim

iFogSim, proposed in [64], is a modular toolkit with key features for Fog simulation, such as power monitoring and resource management, and is initially based on the well-known cloud simulator CloudSim[24]. A *Graphical User Interface (GUI)* allows the user to create the topologies by placing components such as fog devices, sensors and actuators. In this simulator, applications are defined and modelled inside the toolkit, which is useful when studying fog infrastructure but not as much when the focus shifts to application development. In [107], the author extended the work of iFogSim to add a data placement extension to enable users to experiment with different policies.

4.3.3 MiniWorld

Miniworld [144] is an project that proposes an emulation framework similar to the one we propose. It also uses CORE as the network emulator, and provides the possibility of deploying virtual machines and containers as emulated nodes. It, however didn't include mobility models or the integration with other simulators or external tools. Another missing feature was the possibility of running constrained nodes with QEMU, embedded operating systems and host applications. Features that were included in MACE.

4.3.4 Virtual Mesh

VirtualMesh works by creating TUN virtual devices on the Linux host and connecting them to a OMNet++ emulation session. Host applications can access virtual devices as if they were real devices. Hence, any packet created by the Linux kernel network stack is routed to the simulation by a helper application. To do so, the author created a separate scheduler for OMNet++ that works in real-time so that applications can work by following wall time. This scheduler opens UDP sockets in the host, where the help application can connect and send the packets created. This functionality was lately incorporated natively in OMNet++, and in the most recent versions, adapted to use a different approach. The main drawback of this approach is the impossibility of OMNet++ in handling high traffic flow.

[2] By using named pipes.
[3] https://nodered.org/about/

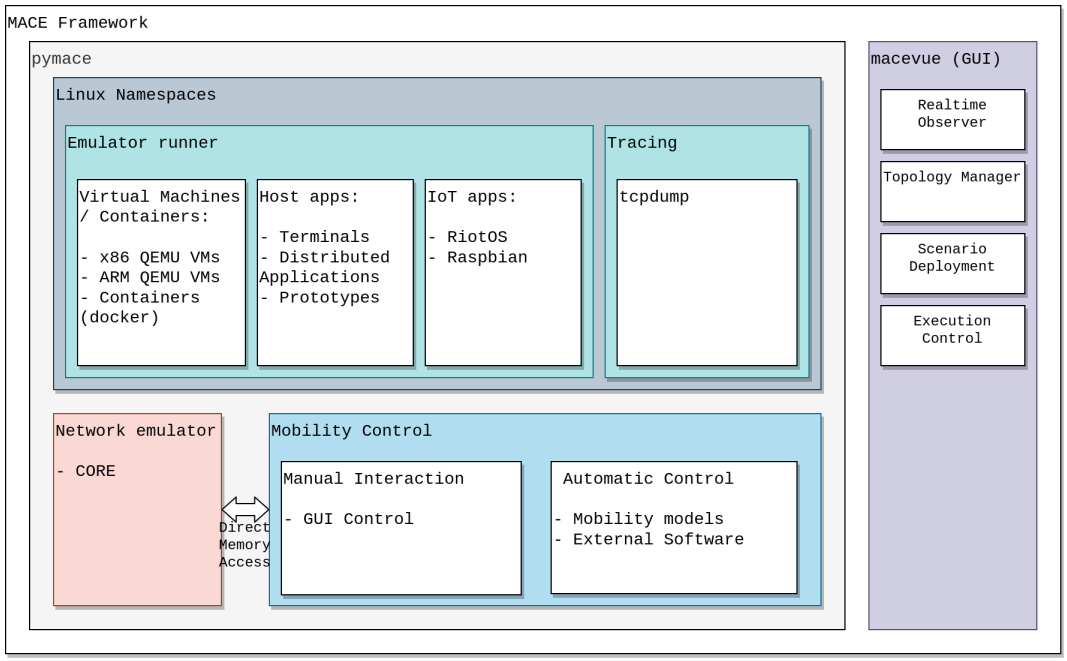


Figure 4.1: Framework Architecture

4.3.5 EmuFog

EmuFog [101] is an open-source project started with the goal of creating a test environment for fog applications. Like the emulator we propose, it uses a network emulator, MaxiNet, to allow the deployment of several nodes in a controlled environment. Applications are emulated in a Docker container, as can also be done in MACE, but lacks the option of adding mobility to the nodes.

4.3.6 Avens

Avens [97] is a framework developed for an older version of OMNet++, that proposes a system where a flight simulator can control the mobility of the nodes. The nodes' mobility is controlled by the flight simulator *X-plane*, and the framework also includes a tool called *Larissa*. The former allows automatic code generation for OMNet++ by using configuration blocks. The main goal of the proposed system is to facilitate the development of new network protocols and technologies for aerial systems by providing a simulation environment. The system lacks, however, the ability to test existing applications as proposed by MACE, and the possibility of using different mobility models.

4.4 Proposed Framework

The proposed mobile emulation framework creates an environment composed of different modules that can be configured according to the need, and Figure 4.1 illustrates the main modules.

4.4.1 Network Emulator

Two different network emulators were considered for developing our emulator, tested and initially provided as options: OMNet++, with the Emulation capability supplied by the INET Framework, and CORE.

OMNet++

It is open-source software written mainly in C++ that is vastly used in the academic world and has good acceptance in wireless simulation studies. The idea of OMNet++ is not to be a simulator, but a framework mainly used to build network simulators. Therefore, it features several libraries that can be coupled to build a network stack that works as a simulator after being compiled. Additional features can be added to OMNet++ through third-party frameworks. The most crucial framework is INET, which among other things, adds the functionality of creating emulated network interfaces when using a real-time scheduler. When using OMNet++/INET emulation feature, the network stack is split so that part of the stack is emulated in the operating system by creating virtual network interfaces, and part is simulated in OMNet++. The rationale behind that decision was to exploit OMNet's ability to simulate different physical environments, propagation media and mobility patterns for the nodes. The kernel built-in option of creating TAP devices is used to create virtual network interfaces on Linux, so that layer two packets can be available to userspace applications.

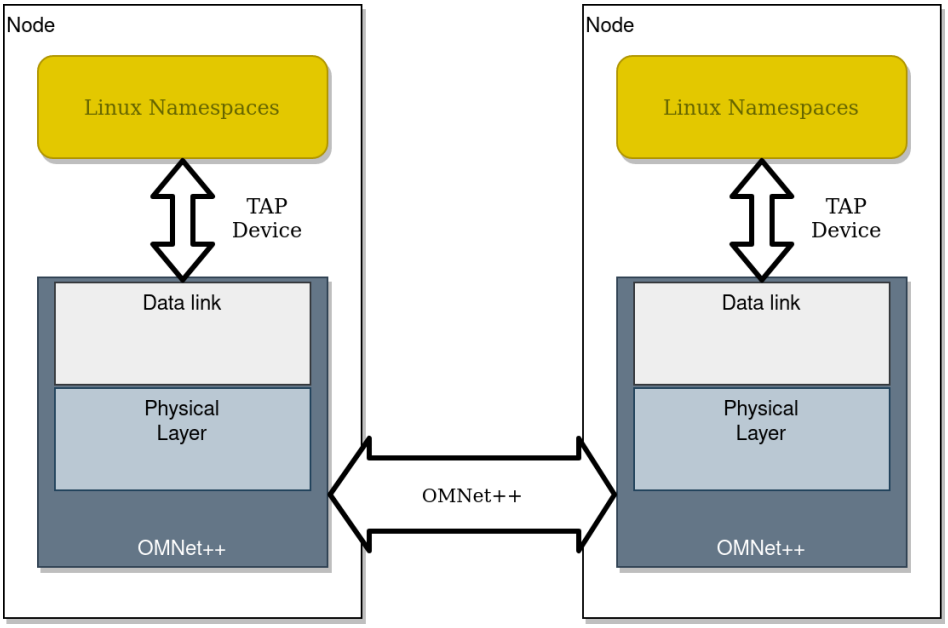


Figure 4.2: OMNet++ Emulation Stack

Applications running on the host operating system see those virtual network interfaces as real interfaces. So, an application created to run in this environment can be transported to an actual network node without changes. Creating several virtual interfaces in the host operating system allows the simulation of what would be several nodes distributed across a network. However, a problem arises when all nodes need to open the same port. Since all interfaces

are running in the same network namespace, when a POSIX socket is open using one port, it owns exclusive access to the port as long as it is open. Running the simulation would require that each process would run the protocol in a different port, but that can be avoided using Linux network namespaces. When using that, each process becomes, at some level, isolated from the others, and they can see only the network interfaces inside their namespace.

CORE

The open-source project CORE works similarly to OMNet++ when it runs emulated interfaces, but the depiction of the wireless interface is less customizable. Additionally, instead of using TAP interfaces, it uses VETH interfaces, which, despite being a different technology, achieves the same results from the perspective of an emulated node. It uses *nftables* to control the connectivity in the physical medium, similar to the Unit Disk Radio module from OMNet++. This means the signal attenuation is not realistically represented, so the Cartesian distance between the nodes defines the connectivity. Figure 4.3 illustrates CORE's emulation stack.

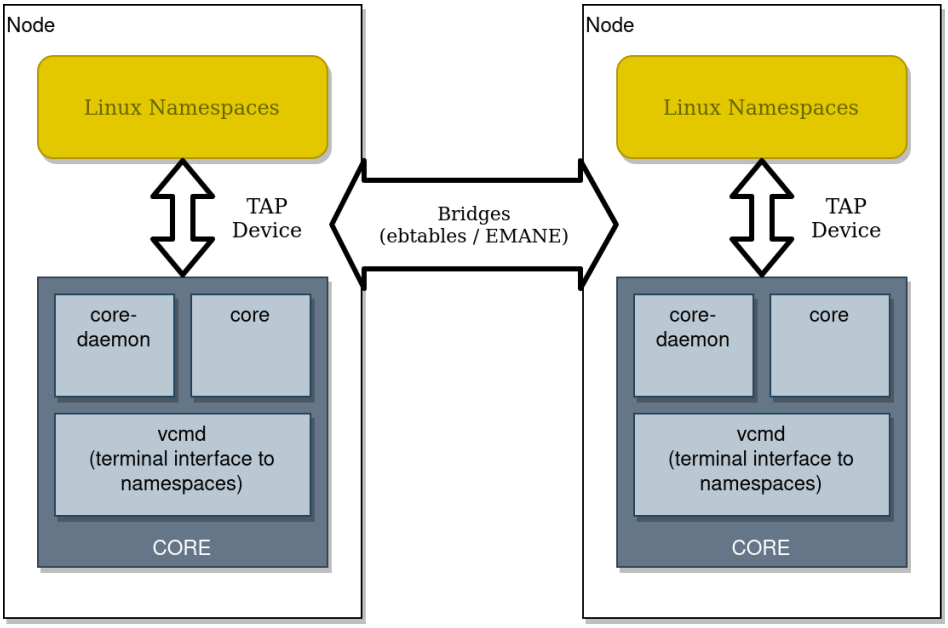


Figure 4.3: CORE Emulation Stack

CORE uses, by default, the solution with Linux namespaces to create a simple container where the applications can be run by making use of the *vcmd* module. All the functions of creating, configuring the nodes, and controlling the traffic are made by the *core-daemon* module. To connect the interfaces that are created inside the namespaces, devices are created outside as their VETH pair counterparts. When the user needs to emulate a more realistic representation of the wireless interface, considering propagation delays and signal decay due to the specific medium characteristics, the user can associate the emulator EMANE [41] together with the CORE emulation session. EMANE focuses on mobile ad hoc networks and has different radio models, such as IEEE802.11abg, *Time Division Multiple Access (TDMA)*,

Long Term Evolution (LTE) and Radio Pipe. It is also an open-source project and is currently maintained by a company called Adjacent Link. CORE session nodes can also be associated with other software to control the routing between the nodes. It is pre-configured to allow the use of the routing suite *Quagga*, but the protocols supplied are not well suited for ad hoc wireless networks.

Routing Protocol

Another essential framework component is the routing protocol, which enables multi-hop communication between mobile nodes. Many different protocols have been created, and each one has an advantage or specific goal depending on the application [114, 4]. Our emulator needed a routing protocol implemented in Linux, which could be used in generic applications and work well with mobile nodes. A comparative study [153] has been made with some of the most prominent protocols for MANETs and FANETs: *Ad Hoc On-Demand Distance Vector (AODV)* [118], *Optimized Link State Routing Protocol (OLSR)* [69], and *Better Approach to Mobile Adhoc Networking (BATMAN)* [14]. BATMAN V was found to have a better packet delivery ratio with mobility and could maintain good connectivity even with *high mobility*. It is also a proactive protocol which can reduce latency since it constantly updates the best route among the nodes.

4.4.2 Mobility Control

The mobility module of the framework is a software capable of controlling the position of each node in the simulation via manual input or automatic control. The automatic control can happen via the implementation of mobility models or an interface to external software. The positions are fed into the network emulator that, based on configurations of the wireless interface's radio range, connects each node taking part in the distributed application. A mobility model must be considered to control the nodes' position in the emulated scenario automatically.

When using CORE as a network emulator, mobility is quite limited. By default, the only option to add mobility is to create predefined scripts associated with each node. They, therefore, loop through this script, creating a repetitive movement. Nodes can be initially placed with the help of a graphical interface supplied with CORE. To avoid this limitation, it has been integrated into the proposed framework, an option to use a Python library[4]. It can be used to control the nodes' mobility so that their position can be injected into CORE emulation session. The following mobility models are included in the library:

- Random Walk
- Random Waypoint
- Random Direction
- Truncated Levy Walk
- Gauss-Markov

[4] https://github.com/panisson/pymobility

- Reference Point Group Mobility model
- Time-variant Community

OMNet++, on the other hand, comes with several mobility models provided by the INET framework, so the user only needs to configure the emulation session according to the pattern more suited to the application. The following models are included out of the box:

- CircleMobility
- TurtleMobility
- GaussMarkovMobility
- FacingMobility
- BonnMotionMobility
- StaticGridMobility
- StationaryMobility
- Spiral
- Drones

Due to the proposed framework's modularity and the fact that CORE sessions can be started from Python scripts, other types of domain-specific simulators can potentially be used to control the nodes' mobility. A proof-of-concept experiment is later introduced in Section 4.5 where a *Uncrewed Aerial Vehicle (UAV)* Flight simulator is used to update nodes' positions in CORE. Other works have also introduced similar functions with OMNet++ [97] [168].

As mentioned before, mobility can also be controlled by an external agent connected to the specific domain of the application. The emulator was connected to an open-source UAV flight simulator to test this ability. Paparazzi[5] is an autopilot developed for fixed and rotary wing UAVs that can be compiled and deployed to many different micro-controllers and autopilot boards. When using Paparazzi, all the UAVs are controlled by the ground station via radio commands. However, it is also distributed with a flight simulator, where the radio link between the UAVs and the ground station is replaced by UDP sockets communicating via pprzlink[6]. The proposed framework also includes a proxy for the *pprzlink* that can capture all packets exchanged between the simulated UAVs and the ground station, acting as a man in the middle. As a result, the emulator can capture in real-time the actual GPS position of the UAVs and convert it to a position in the emulated topology.

[5] https://wiki.paparazziuav.org
[6] A communication protocol developed for serializing Paparazzi communication.

4.4.3 Distributed Mobile Applications

Running emulation scenarios instead of simulated applications created inside simulation frameworks creates opportunities to run distributed mobile applications in more diverse scenarios. That consequently increases the ability to create heterogeneous sessions by running some nodes directly on the host and others inside virtual machines or containers, as shown in Figure 4.4.

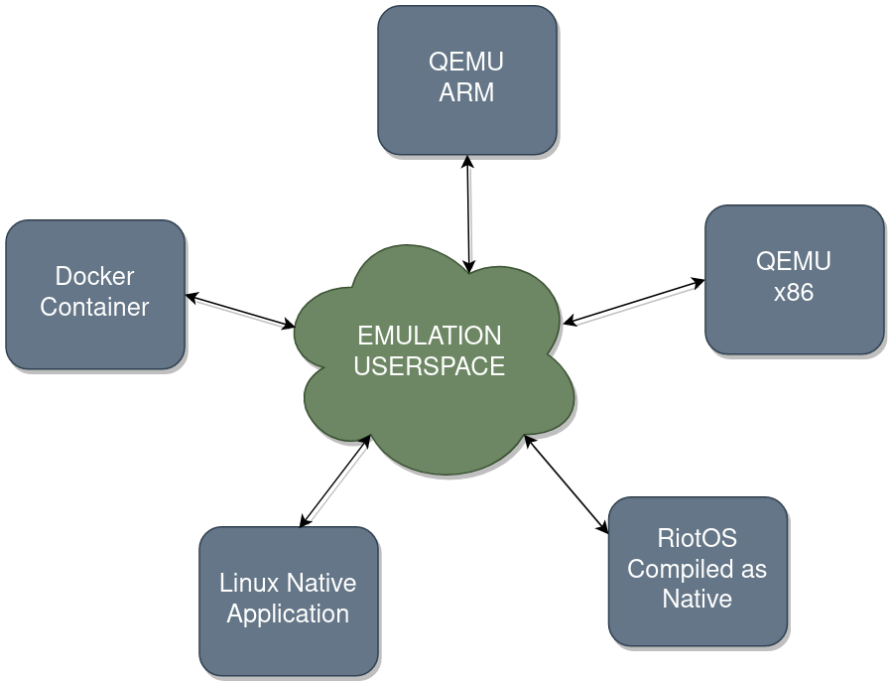


Figure 4.4: Emulation Options

To achieve such scenarios, one can run the same protocol in different operating systems using virtual machines. Each VM can be tuned with different hardware definitions, such as available persistent and volatile memory or the number of processor cores. Furthermore, it is also possible to mix nodes running in different architectures, such as x86 and ARM. The same is true when running native applications with others made for different operating systems, such as Riot OS or other embedded OSes that can be compiled as a host application for testing. The sections below describe how each option is implemented in the framework.

Linux Native Applications

When running applications inside Linux namespaces, due to the isolation, they can open sockets using the same port number. Hence, it is possible to run any application that accepts more than one simultaneous process instance. That allows, for example, testing several instances of web servers, distributed file systems or any application. An evaluation environment was created to run a mobile distributed experiment with *etcd*, and more details can be found in Section 4.6.1. When it is necessary to increase the isolation level, the application can be run as a VM or a container. That adds the drawback of consuming more resources from the host

computer and, consequently, allowing the deployment of fewer emulated nodes.

Running Virtual Machine Nodes

Instead of running distributed applications directly in the host operating system, it is also possible to run guest virtual machines (VM). The framework allows running a VM in each node where the virtual network interfaces will be bridged to the emulated one. The framework provides a simple way of managing the pre-configure VMs images, which can be x86 or ARM for embedded devices. The overall architecture can be observed in Figure 4.5.

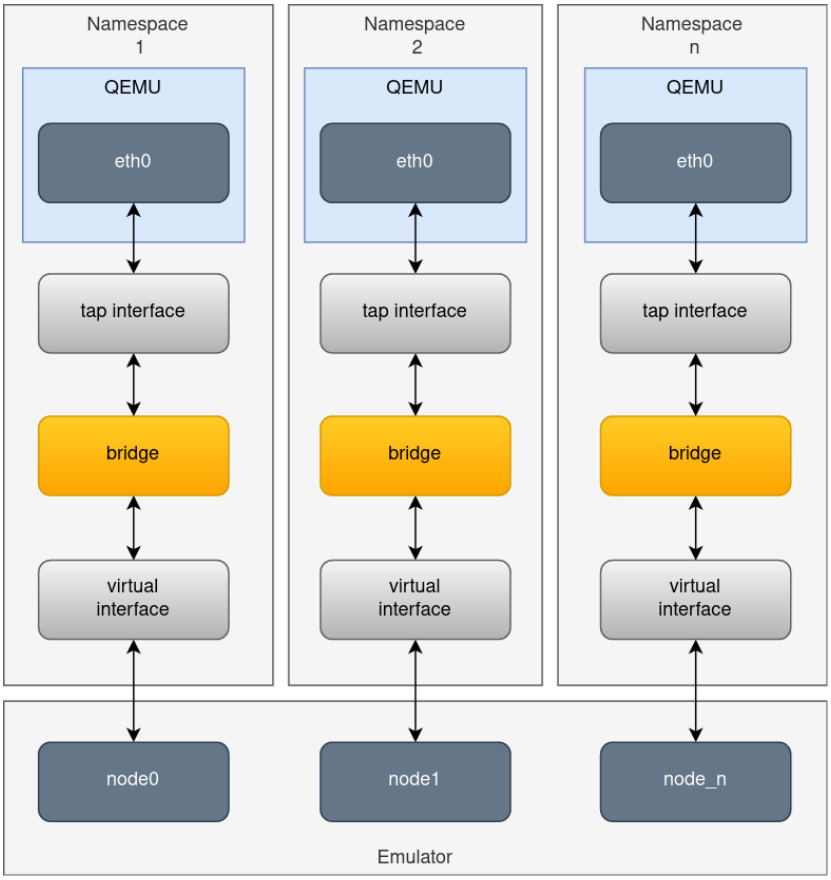


Figure 4.5: Virtual Machine Architecture

Besides being able to choose VMs to run in each node, it is also possible to choose Docker containers, therefore consuming fewer resources from the host machine.

Running Embedded Operating Systems

A critical step of developing distributed algorithms for heterogeneous IoT devices is the ability to test them on different operating systems and hardware with unbalanced performance. Nowadays, many embedded OSes are available, and RiotOS[7], for instance, has active development and commitment to keep a reduced footprint whilst enabling the latest technologies

[7]https://riot-os.org/

to be implemented. RiotOS allows compiling the firmware as an x86 (native) application for testing. It can run with the wireless interface being emulated by a TAP interface, which the framework can bridge to an emulated interface. With that, one can test the communication between the emulated firmware and other nodes in the system, such as an edge server running on a virtual machine in another node.

4.5 Evaluation of Network Emulators

Since the network emulator is an essential component of the framework, some tests were run to assess the maximum performance that can be expected from them and build a baseline for further assessments. For that, benchmarks were taken with *iperf3*[8]. The reasoning was to inject different traffic flows into a topology and measure the throughput and losses. All the tests were performed on the computer described in Table 4.2.

Table 4.2: Test Platform

Processor Manufacturer / Model:	AMD / Ryzen 7 1700
Number of cores	8 (16 Threads)
Cache L1,L2,L3	768kB, 4MB, 16MB
Memory	16GB DDR4
Emulator	OMNet++ 5.6.1 and CORE 7.2.1
Operating System	Linux Mint 20 x86 64bits
Routing Protocol	B.A.T.M.A.N V[9]

The experimented topology is composed of nine symmetrically displaced nodes with an equidistant one-hop distance between each other. Figure 4.6 illustrates the topology. An emulated wireless ad hoc network with the parameters set as illustrated in Table 4.3 was created in OMNet++ and in CORE.

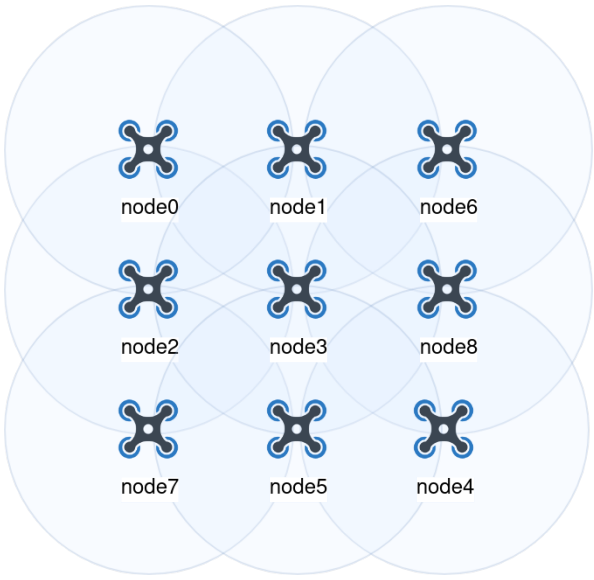


Figure 4.6: Symmetrical topology

[8]https://iperf.fr/en/

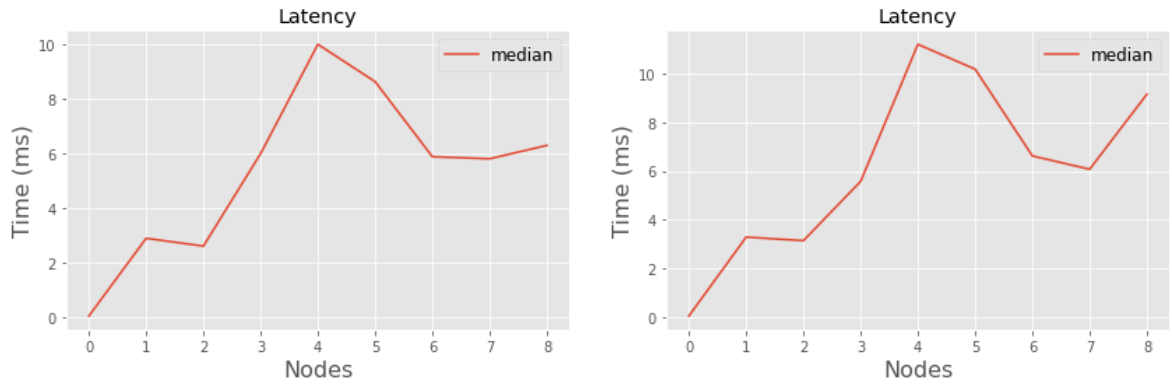
Table 4.3: Emulation parameters

Setting	OMNet++	CORE
MTU	1500B	1500B
Interface	ExtUpperIeee80211Interface	N/A
MAC	Ieee80211Mac	N/A
Management	Ieee80211MgmtAdhoc	N/A
Op Mode	AC	N/A
Bitrate	433.3Mbps	433.3Mbps
Number of antennas	6	N/A
Delay	N/A	1300us
Jitter	N/A	5 us
Error rate	N/A	1

Below, the results for latency and throughput are presented. These establishes a baseline for what can be expected as maximum practical data throughput for any application running in similar conditions.

Latency

For the latency, the measurements were taken as a round trip between one node in the corner[10] and all other nodes in the tested topology. Ping was used since it is a trivial and effective way and adds only a small overhead when configured with a small payload. For the *basic range model* used in CORE the latency is pre-defined in the configurations, so its value was set to match a similar latency achieved by the OMNet++ emulation illustrated in Figure 4.7a. The median latency measured when using CORE was similar to the designed, as can be seen in Figure 4.7b. This way, the values would be more comparable when measuring the throughput.



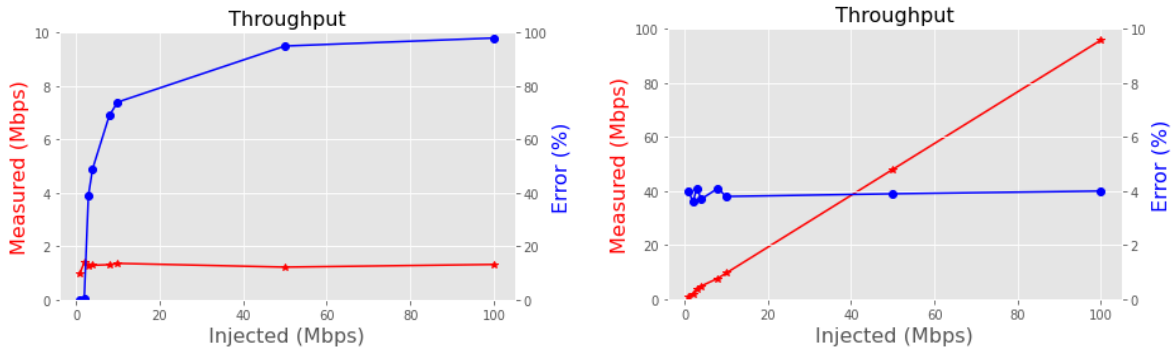
(a) With OMNet++ as Network Emulator (b) With CORE as Network Emulator

Figure 4.7: Measured latency from one Node as Reference

[10]Since they are equidistant, any corner node should yield the same results.

Throughput

For measuring the throughput, the worse case was considered. Flows were created traversing the maximum distance available in the topology between the nodes in the diagonal opposing corners.



(a) With OMNet++ as Network Emulator (b) With CORE as Network Emulator

Figure 4.8: Measured throughput from one Node as Reference

When using OMNet++, the real-time scheduler needs to keep up with processing all the packets in the whole topology. Furthermore, the scheduler must run in only one thread, and as seen in Figure 4.8a, when there are too many packets, it cannot keep up. Even when injecting throughput below the theoretical expected based on the configuration, it can deliver less than 2Mbps. So, all injected flow beyond this threshold is ultimately dropped, with losses reaching almost 100% when injecting 100Mbps. Conversely, CORE could keep up with the injected flow whilst keeping the error rate fixed in the configured value of 1% per hop, as can be seen in Figure 4.8b. Using OMNet++ for emulation sessions seems disadvantageous compared to CORE, unless the user needs to study the detailed behaviour of different radio technologies or medium access protocols, for example.

4.6 Experimental Evaluation of Distributed Mobile Applications

4.6.1 Emulation tests with etcd

etcd [42] is a distributed key-value store with broad acceptance among system designers and is used by tools largely used in production, such as a configuration service for Kubernets. It provides strong consistency among the nodes by using Raft as a consensus protocol and can be used by any application that requires a consistent key-value store. For this proof of concept, the proposed framework is used to create an emulation session with the designed topology, and etcd is deployed in namespaces using ramdisks as storage units. Etcd has available on their website baseline performance figures[11], alongside a benchmark tool that can run in any deployed system. Since the traffic flow when testing etcd is too high, only CORE could be used, and the testing platform and settings are the same already shown in Table 4.2 with the

[11] https://etcd.io/docs/v3.4.0/op-guide/performance/

topology presented in Figure 4.6. The tests were performed with the same parameters stated on their website and shown in Table 4.4.

Table 4.4: etcd Parameters

Key size	8 Bytes
Value size	256 Bytes
etcd version	3.4.13
Fixed leader	Node 3

Table 4.5: etcd Results

1 con. 1 cli.	Baseline	1300ms	300ms	300ms mob.
Average QPS	583	231	567	40
Average Latency	1.6ms	43ms	18ms	24ms
100 con. 1000 cli.	**Baseline**	**1300ms**	**300ms**	**300ms mob.**
Average QPS	44 341	7 184	11 172	4432
Average Latency	22ms	137ms	85ms	224ms

The results are below the baseline, which is expected considering the high latency configured in CORE for the links. Reducing the latency to 300ms instead of 1300ms, increased the average queries per second to 11172, and the average latency was reduced to 85ms. Mobility was then added with the random walk model, where nodes could move freely in a square area. As seen in Table 4.5, with mobility, there is a considerable decrease in performance, with lower throughput and higher latency.

4.6.2 Distributed key value store

A key-value store was designed, verified and tested to verify how the proposed framework helps in prototyping new distributed applications. The application uses a multipaxos log as the consensus algorithm and has an eventual fault detector handling leader election.

The applications were emulated using the same configurations and topology as the etcd emulation described in Section 4.6.1. For the tests, one client was connected to the endpoint of the leader 200 write operations were performed with a key of 8 bytes and value with increasing values. The throughput and latency results are presented in Figure 4.9.

Therefore, a designer can prototype new mobile distributed applications and run existing ones whilst having control of the nodes' position in real-time. Being able to state of the art applications is valuable when creating prototypes to have reproducible benchmarks.

4.6.3 Various Consensus Protocols with Mobility

As mentioned before, running quorum-based consensus deployed in dynamic networks pose challenges due to the fallibility of the communication links. When designing the most common protocols, some link failures are usually expected, but not with the frequency observed in swarms of CPSs. In [3], the authors evaluated the performance of different multipaxos

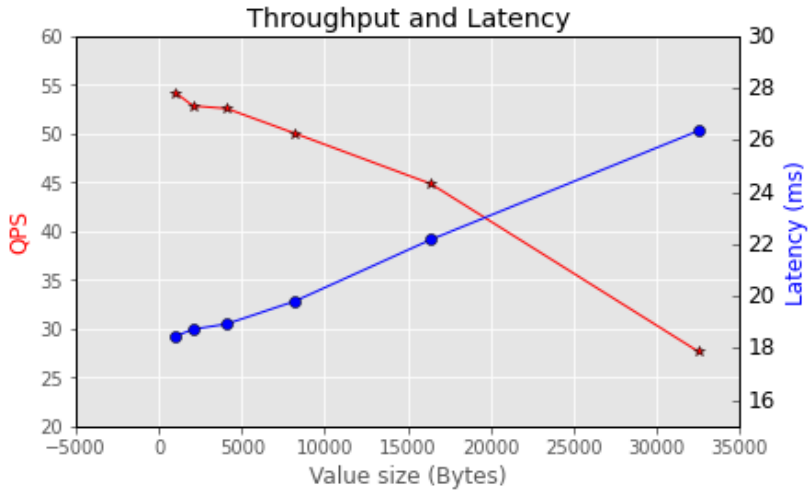
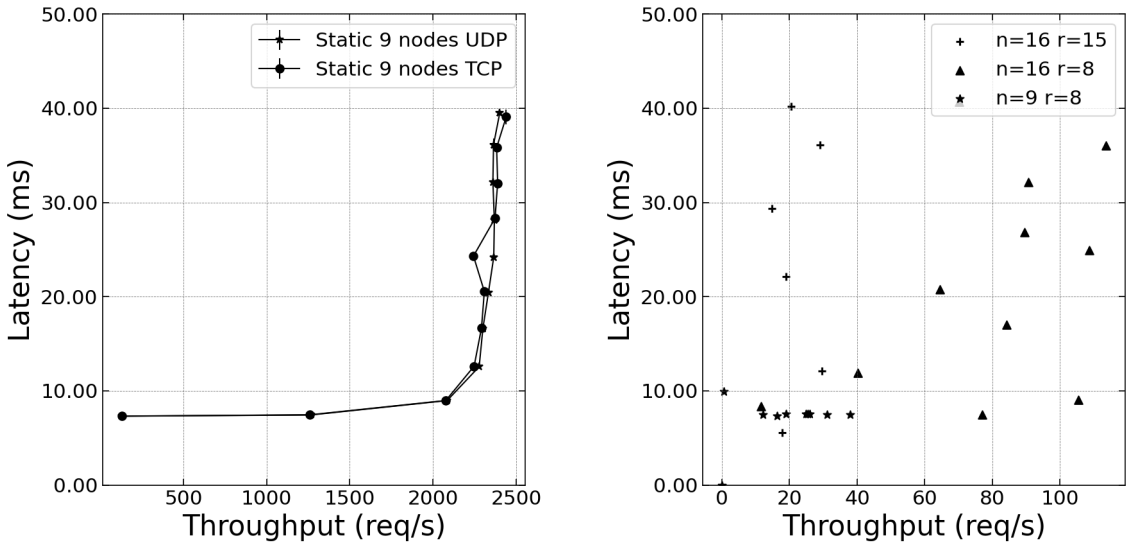


Figure 4.9: Key Value store Results

algorithms and made freely available a library with the implementation of those algorithms. Therefore, we introduce here some benchmarks run the *paxi* library and MACE, deployed in the context of swarms of CPSs. The links were configured for these tests with 433Mpbs, 1000ms of delay, and no error or jitter. The *paxi* library was configured as per [3].



(a) Static with 9 Nodes

(b) Dynamic with 9 or 16 Nodes

Figure 4.10: Saturation of Paxos Protocol with Fixed Topology and TCP and UDP Transport Protocols

Figure 4.10a shows the results when the topology is fixed, and the transport protocol used is TCP and UDP. As mentioned in [3], the leader-based consensus protocols eventually create a bottleneck in the leader node due to the excess of messages to process when compared to

the other nodes. When considering an Ad hoc network, this issue is eventually also present in the network layers for the nodes surrounding the leader since they serve as a route for packets. This effect can be visualized in Figure 2.4 in Section 2.2.4, where it is observed the higher number of packets on the leader node. A significant drop in performance is observed when mobility is introduced, as shown in Figure 4.10b. In this particular case, the nodes can move with the random waypoint model across an area of 300 by 300 meters, with speeds between 1.5 and 3 meters per second. As mentioned in Section 2.3.2, by increasing the density of nodes, the messages have a higher probability of reaching the destination, which helps increase the average performance. Also, by increasing the number of nodes, the quorum required for Paxos to maintain liveness is higher. So the best performance was observed when having more nodes, but not configuring all nodes as replicas. The communication complexity of Paxos has a bigger negative impact the higher the number of replicas, so a trade-off must be considered between safety and performance. These tests were run with the configuration of an adaptive leader, which means that the leader remains fixed throughout the simulation for the Paxos protocol.

4.7 Future work and Limitations

One of the main observations in this work was the issues when running OMNet++ as the network emulator. When increasing the number of packets, the emulator cannot keep up with the traffic even though the computer processor usage does not reach its limit. That could indicate that the code can be enhanced to achieve better results. Another observation was that when using CORE, the BATMAN routing protocol requires more than 20 seconds for the routes to converge so traffic can start flowing. When using OMNet++, on the other hand, it requires less than 2 seconds.

In future improvements of the MACE framework, new functions useful when experimenting with Edge/Fog infrastructure are still to be added: cost model, automatic application migration and energy model. Also, to increase scalability, we plan to explore CORE's feature of running distributed emulation sessions.

4.8 Conclusions

In this chapter, we introduced the MACE emulation framework, which is aimed at Edge/Fog application development, focusing on Mobile Ad hoc Computing. It was built on top of existing network emulators and introduced new features that enable fast application prototyping, instrumentation and emulated deployment. The main functionalities were demonstrated by experimenting with an off-the-shelf distributed configuration service and a prototyped distributed key-value store and showed how an emulated environment could be deployed to enable reproducible research with mobile distributed computing. An emulation tool allows the user to use the paradigm of *coding only once* during the development and research cycles. The same prototype code can be tested under realistic scenarios, even simulating a MAC protocol. The prototypes can be deployed on a mobile multi-node platform that otherwise would require time-consuming tasks of firmware updating. It is, however, impacted by the side effects of emulation, such as execution in wall-time, and competition for the computer

resources by the several components. Most of these side effects can be overcome by using fast simulators, such as the one we introduce in Chapter 3. For future work, we plan to leverage the built-in functions for distributed emulation available in both CORE and EMANE. This emulation framework is open-source [22] and in constant development.

Part II

Applications of Distributed Algorithms in the UTM Context

Chapter 5

Consistent Position Tracking System for Very Low-Level Airspace

Contents

5.1 Introduction

Recently, there has been increasing interest in UAS for many real-life, civilian applications [148], including real-time monitoring, remote sensing, search and rescue, agriculture services and delivery of goods. In addition, a rapid growth in the number of drones in activity is expected, especially in very low-altitude airspace with high demand in densely populated areas [151]. According to [135], European Union´s SESAR predicted fully autonomous flights to start in 2025, while in 2035 autonomous flights with passengers. The authors also mention a forecast that by 2035, 98% of aircraft operating in Paris airspace will be autonomous. Unmanned aircraft can still have a pilot located in *Visual Line of Sight (VLOS)* to guarantee safety in case of conflicts with the aircraft original path. To allow flights BVLOS, an increased level of safety and automation is required and currently, when allowed, they still require a pilot in command. The current *Air traffic management (ATM)* system is not enough to cope

with the requirements for autonomous UAS flights. That highlights the urgency of a UTM system capable of dealing with autonomous flights whilst delivering the adequate level of safety.

One of the key components of the emerging UTM system is the capability to track the vehicles' positions. Currently, tracking and position reporting of aircraft relies on long-range radio communications. Unfortunately, a large number of vehicles communicating through conventional RF in densely populated areas is likely to incur mutual interference and, consequently, poor performance. One way of reducing interference is to limit the range of the wireless radio, which in conjunction with the limitations introduced by the buildings, can render systems such as *Automatic Dependent Surveillance–Broadcast (ADS-B)* unfeasible. Furthermore, it remains unclear how the current ATM systems could be redesigned to enforce highly available, consistent position reporting to meet safety and performance requirements of novel UTM systems.

This study aims to investigate the feasibility of a novel tracking and position reporting system that could replace ADS-B in the context of UTM and UAS. The rationale behind it is two-fold. First, the new system should allow small autonomous vehicles to communicate with low-cost, low-footprint, short-range wireless interfaces and broadcast real-time data with local stations in the vicinity. Second, the system should rely on distributed computing techniques to share positioning data of vehicles with high availability and strong consistency, allowing even aircraft beyond line of sight to have updated information, ensuring proper functionality of their own applications.

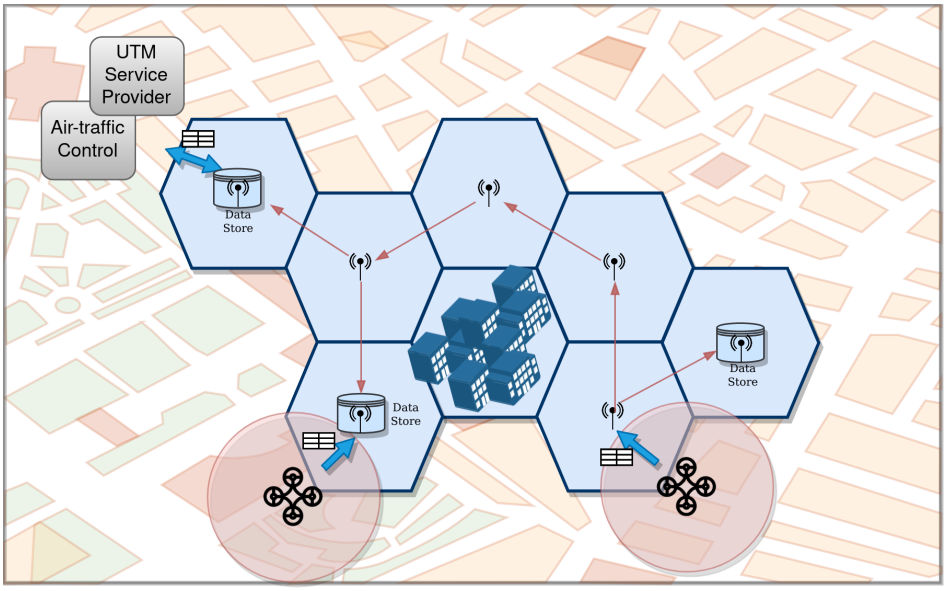


Figure 5.1: Distributed UTM Data Service

In this Chapter, we introduce a system architecture to reach the aforementioned design goals by deploying a distributed tracking and position reporting system throughout Radio Access Network cloudlets stations, as depicted in Figure 5.1. The architecture provides performance requirements, such as low-latency, high-throughput data exchange, thanks to 5G's communication capability. To meet the requirements of a consistent and highly available po-

sitioning data sharing, our architecture leverages the functionalities of a strongly consistent distributed key-value store. Preliminary results based on mobile systems' emulation suggest that this architecture meets both performance and safety requirements of emerging UTM's tracking and position reporting system.

5.2 State of the Art

5.2.1 Unmanned Traffic Management

The rapid expansion of the market of small UAS has created new risks for civil aviation [94][135]. That increases the need for regulation and deployment of a fully functional UTM [40]. Some governmental agencies have been more prominent in UTM proposals, such as the USA with NASA/FAA and European Union's SESAR/U-space. In [76], the author described NASA's research initiative related to the concept of operations (ConOps). NASA established a research platform which is used together with their partners to test and evaluate the challenges and solutions proposed for each of the technical capabilities associated with UTM. For NASA/FAA the UTM is set to evolve incrementally according to *Technical Capability Levels (TCL)*, and some research works have been conducted in order to verify the full development and verification of TCL levels. In [132], flight tests demonstrating the most basic TCL levels 1 and 2 were performed even with BVLOS for unpopulated areas. BVLOS in populated areas is introduced in [131], where the authors performed a flight demonstration of TCL3. In [26], the author demonstrates TCL4 capabilities using simulation tools and emphasises safety by describing in detail the SAFE50 architecture. U-SPACE/CESAR Concept of Operations [119], as shown in Figure 5.2, has a level system similar to NASA/FAA that increases with the technology capability of supplying high levels of automation, but characterized by the services that the UTM system can supply. The foundation services (U1), being related to infrastructure, such as geo-fencing, registration and identification. Meanwhile, the full services (U4) must supply integrated interfaces with manned aviation, rely on very high levels of automation, connectivity and digitalisation for both the aircraft and U-SPACE systems.

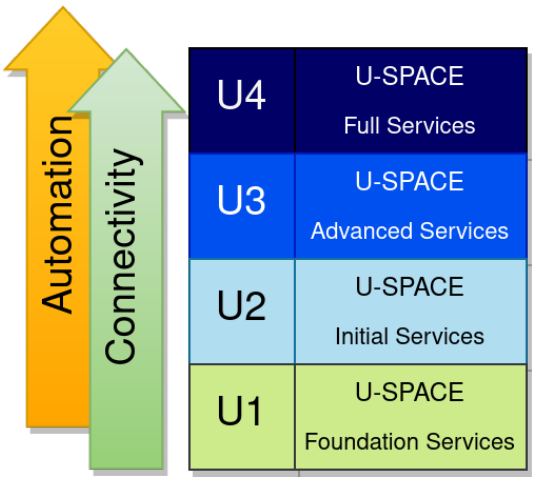


Figure 5.2: U-SPACE Service Levels (based on [119])

The framework on how a UTM system should work and the interaction between the stakeholders is not yet finalized [102], and many aspects are not clear in order to achieve the highest levels of autonomy. However, the main proposed architectures are composed of distinctive applications [80] such as sense and avoid [147, 104, 116, 178], path planning [17, 162, 161, 27], which rely on the fact that trustable data can be shared among stakeholders with low latency. And NASA/FAA have specified some of the high level functional blocks of a UTM system, as represented in Figure 5.3, where it is possible to identify the main stakeholders, and how they interact with each other.

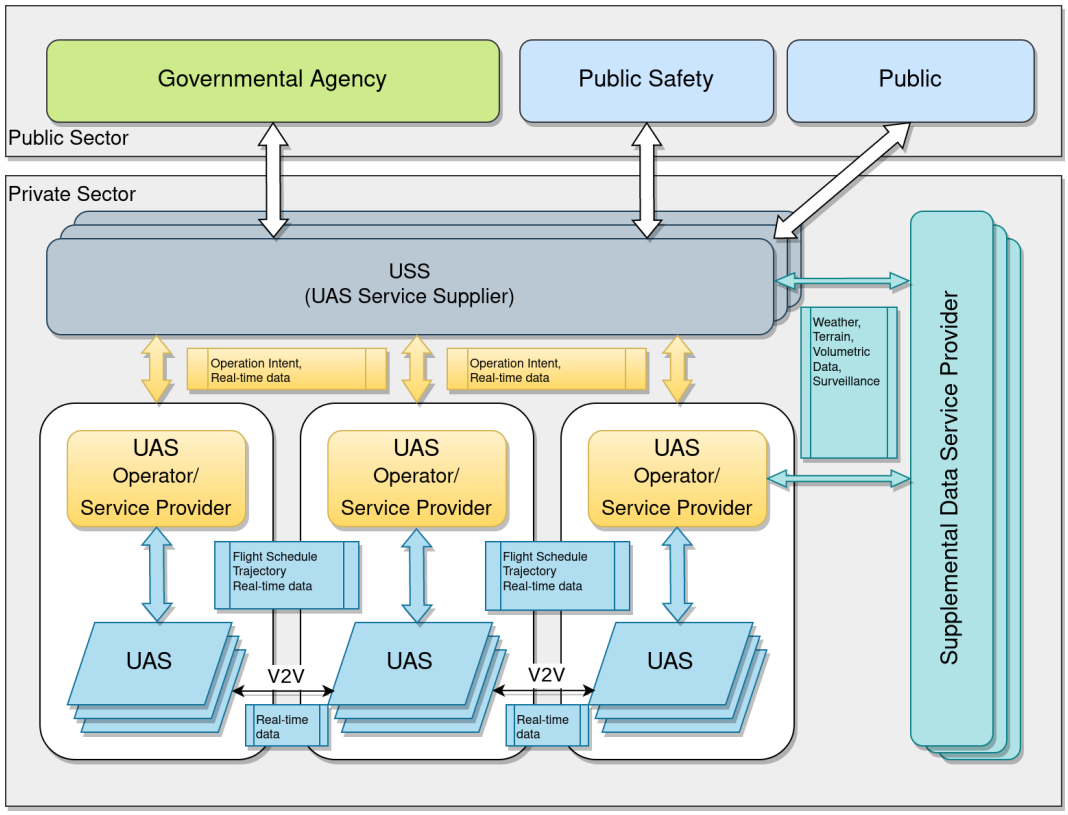


Figure 5.3: UTM - Architecture based on FAA and U-SPACE

5.2.2 UTM Stakeholders

As detailed in [102] and [135], the UTM is comprised of different stakeholders, each of which with specific interests and capabilities.

UAS Service Supplier

The service supplier is an entity that might or not have commercial interest, and helps maintaining a safe use of the airspace in which it operates. As seen in the architecture in Figure 5.3, it acts as intermediator between the local authorities, i.e a governmental agency, and the UAS operators. It is expected that one or more *UAS Service Suplier (USS)* shall operate in the same airspace, sharing information on a decentralized way, whilst aiming at maintaining

the reliability of trusted oracles. Such characteristic is necessary to avoid Sybil attacks [177] on the system that aim in providing unfair competitive edge to a stakeholder.

UAS Operator

The UAS operator is the owner of a fleet of aircraft and is responsible for its safe operation. It, by supplying origin and destination for its missions, requests the USS for flight schedules and trajectories. It is responsible for sharing real-time data related to its aircraft's position, velocity and state.

Local Authority

Currently, different UTM solutions are being developed with the goal of supplying basic services for a expected boom of UAS traffic in major cities and urban agglomerations. One of the challenges in deploying a fully autonomous UTM system is a deconfliction system, which differs from the current manual deconflict system adopted by ATM. Moreover, deconfliction can be further decoupled from the first stage proactive deconfliction related to flight planning and scheduling prior to take-off. All UAS must be equipped with a sophisticated, fast, *Sense And Avoid (SAA)* system capable of running either centralized, on a UTM system, for major route planning and deconfliction, or decentralized with components embedded in each aircraft [152]. The SAA has usually dual features, which are often referred as Global and Local planning. Global since a system would have a trustworthy view of the whole system and would be able to create a conflict-free flight plan that would ensure that during the plan execution, there would not a volumetric collision with a known obstacle or aircraft, with a certain degree of confidence. However, unforeseen obstacles or rogue aircraft could be introduced along this pre-defined plan, leading to the need of a local SAA system running in each aircraft capable of detecting potential collision, and leaving a feasible reaction time for the aircraft control system to evade the conflict and later return to the pre-defined global plan.

5.2.3 Aircraft Surveillance

A position tracking and reporting system is a crucial system for aircraft surveillance. The ADS-B, illustrated in Figure 5.4, is a service present in aircraft[1] that broadcasts relevant unencrypted data in real time. It aids radars in keeping updated and correct information about all the aircraft currently flying. With such information about all aircraft, ATM authorities can develop applications to improve the security and efficiency of air transportation systems and airports. Albeit not mandatory like *ADS-B Out*, other aircraft can also receive broadcasts with *ADS-B In* and use the information to build knowledge about its surroundings in order to improve conflicts avoidance. Aircraft equipped with ADS-B equipment periodically broadcast their call sign, latitude and longitude, ground speed and flight number with signals modulated in 1090 MHz.

[1] Mandatory in many countries depending on aircraft weight or airspace class.

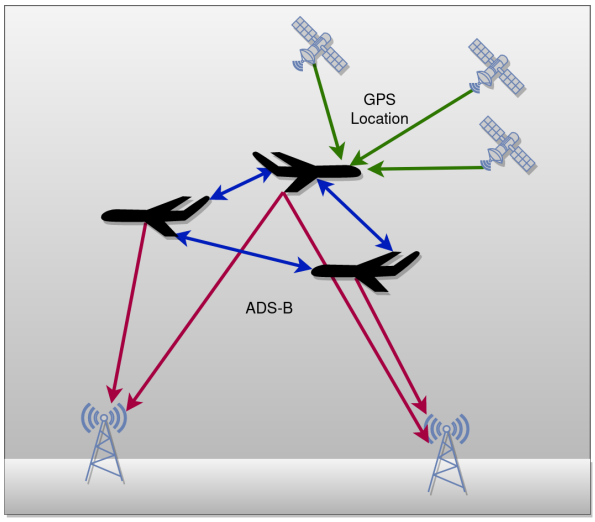


Figure 5.4: ADS-B System Overview

Considering the importance of systems like ADS-B have for ATM, a similar technology is also expected to play an essential role in UTM. Unlike large aircraft, small UAVs are harder to be detected with radars, increasing the importance of a *ADS-B like* system. Previous works have proposed ADS-B alternatives in the context of UTM as explored in [112]. The authors in [88] and [89] proposed an architecture based on low-cost radio transferring data to a central cloud, which does not solve the problem of fast replication across the urban area for low latency and availability to all stakeholders. In [89] the author proposes an ADS-B like system for UTM based on *Automatic Packet Reporting System (APRS)*, which is an approach different to what is proposed in this work since it is based on relatively long-range radio (40Km) with restricted throughput. In [104], the author proposed a solution to inject the position data in the SSID of WiFi, and the results have shown that the proposed setup yielded reasonably low latency (under 125 ms) and acceptable throughput (4 messages/s) under an experimental setup. By having a secure ADS-B like data layer, applications such as presented in [178], where the authors propose a sense and avoid system that used ADS-B data as input, can be built.

5.3 A Novel System Architecture for Tracking and Position Reporting

Our system architecture is designed to enable fault-tolerant and efficient data sharing for UTM stakeholders. The main blocks of this architecture are shown in Figure 5.5. It is built on top of two key components: a distributed, strongly consistent key-value store and the next-generation cellular communication technology. The distributed key-value store is composed of distributed UTM Data Service (UDS) units spread across the urban area, as shown in Figure 5.1. We assume that UDSs are deployed in federated cloudlets interconnected through a low-latency mobile network, such as 5G or 6G[9]. In this section, we describe the design of our architecture by focusing on the *State Machine Replication (SMR)* protocol and the architecture's API endpoints. We skip further details about the mobile network due to space

constraints.

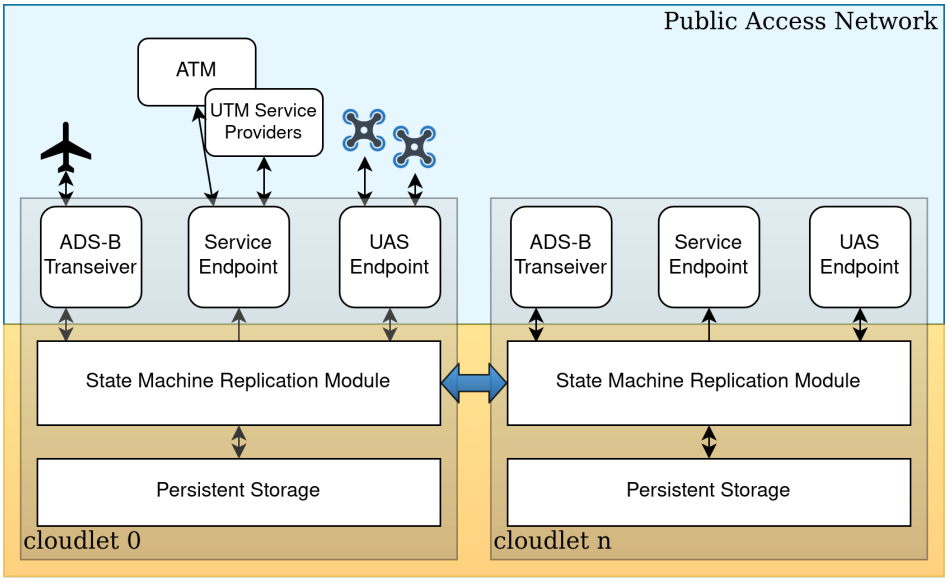


Figure 5.5: Our System Architecture

5.3.1 State Machine Replication

To enable fault-tolerant, highly available key-value store, UDSs run SMR protocol[145]. SMR is a technique to implement fault-tolerant service by replicating servers and coordinating client interactions with server replicas. In particular, stakeholders (e.g., service providers, operators and vehicles) are clients, and servers/replicas are UDSs. Our architecture leverages *etcd* [2], a distributed, strongly-consistent key-value store, to run SMRs. This distributed approach solves important problems of ADS-B [74], such as lack of fault tolerance, data inconsistency, and unpredictable availability of shared data.

For this proof of concept, the proposed framework is used to create an emulation session where etcd is deployed in namespaces using ramdisks as storage units. Etcd has available on their website baseline performance figures[3], alongside a benchmark tool that can run in any deployed system. Since the traffic flow when testing etcd is too high, only CORE could be used, and the testing platform and settings are the same already shown in Table 6.1 with the topology presented in Figure 4.6. The tests were performed with the same parameters stated on their website and shown in Table 4.4.

5.3.2 API Endpoints

The API endpoints are used to put or get data from the system.

- ADS-B Transponder - A ADS-B receiver connected to the system can put data from aircraft.

[2] https://etcd.io/
[3] https://etcd.io/docs/v3.4.0/op-guide/performance/

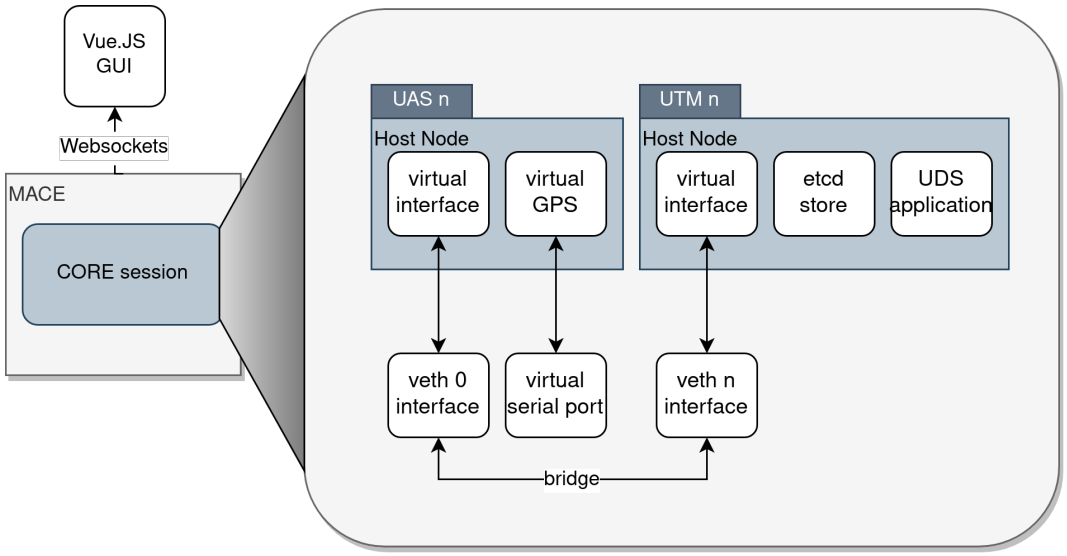


Figure 5.6: Emulation Architecture

- Service Endpoint - The end-point that can be used by the UTM service providers or operators to get the current and past state of the airspace via restful API, time series or stream.
- UAS Endpoint - Used by UASs to insert broadcast data into the system.

5.4 Experimental Evaluation

This section presents the performance evaluation of our system architecture using MACE [46], an emulation framework to study, design and evaluate mobile Ad-hoc applications. Figure 5.6 shows the architecture of our emulation with MACE, detailing how the main blocks communicate.

5.4.1 Cloudlets and Aircraft

MACE uses CORE as a network emulator, emulating each network instance as Linux namespaces serving as minimal containers. Each aircraft client application runs inside these namespaces, and they communicate via *veth* network interfaces with connectivity controlled by the distance between the nodes. The cloudlets are also emulated in such namespaces with a server running the UAS endpoint and interfacing instances of etcd running in the same namespaces. The emulated scenario is run over an area of one square kilometre.

5.4.2 Mobility

The *Random Waypoint* mobility model was adopted for the experiment. In this model, each aircraft receives a random waypoint and a random velocity to simulate a mission's objective. The movement of the aircraft is emulated in MACE, and the real-time position is injected

directly into the network emulator so that it is reflected in the network connectivity. The position is made available to the applications running in the virtual aircraft via UNIX sockets.

5.4.3 UAS Broadcasts

For the payload reporting, a client running in each aircraft broadcasts a JSON object containing the position and additional data via IPV4 UDP sockets using the emulated ad hoc wireless links. The payload also includes a unique message ID, timestamp, aircraft ID, velocity and status. The unique message ID is used to verify that the message was received, and the timestamp is used to calculate the latency between the broadcast and full replication.

5.4.4 UDS Cloudlet

A server representing them is deployed in each emulated cloudlet to simulate the UDS cloudlets. The servers within range of a broadcasted message will receive the new data in the UAS endpoint and replicate it to the other towers that are not in range by using the state machine replication layer. Data is also written in persistent storage for posterior analysis. The consistent replication ensures that the data is reliable across all participant cloudlets so that distributed third part actors can see this reporting system as a trusted oracle for UTM applications such as asset managers, schedulers and collision avoidance path planners. When a message is fully replicated among all replicas, a callback function records the time stamp to calculate the total latency.

To evaluate our architecture, a server representing a UDS was created for each emulated cloudlet. In contrast, a client that broadcasts data via Ad-hoc wireless links using UDP packets represents an UAS. The servers store the information on a *etcd* server, which replicates the data to the other cloudlets. The clients can communicate only with cloudlets in their radio reach, and the cloudlets can communicate with each other via a multi-hop routing protocol. All our experiments were performed on a Linux server whose configuration is described in Table 5.1. Our empirical evaluation is split into two scenarios detailed in Table 5.2.

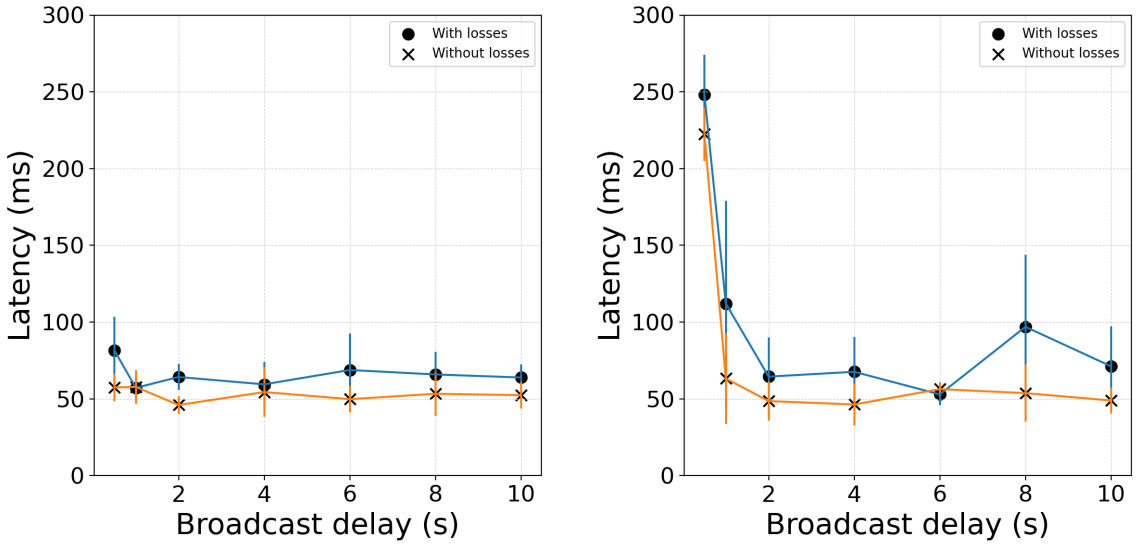
5.4.5 Preliminary Results

In the first scenario, the main goal was to measure the latency between the data broadcasting and full replication when there are no network losses between cloudlets or in the UAS communication interfaces. One of the side effects of the strong consistency is the communication complexity consequence of the multistep consensus algorithm. So, in addition to the broadcast latency, a component corresponding to the replication latency is also expected.

Table 5.1: Test Platform

Processor Manufacturer / Model:	Intel i7-8550U @ 4GHz
Number of cores	8
Memory	16GB DDR4
Emulator	MACE
Operating System	Linux Mint 20 x86 64bits
Routing Protocol	B.A.T.M.A.N IV

We assume that ADS-B broadcasts two messages per second since detect and avoid / path planning algorithms usually run with frequencies ranging from 1Hz to 2Hz. A previous experiment with only one fixed UAS was run to measure the replication overhead. Running such a scenario yielded a replication overhead of 10ms, which is acceptable but is closely related to the number of replicas and, more considerably, to the total coverage area. Larger areas would require more communication hops even if the number of replicas is left unchanged, hence increasing the total latency.



(a) 10 Replicas and 5 UAS

(b) 10 Replicas and 20 UAS

Figure 5.7: Total replication latency with increasing broadcast delay

In the second scenario, losses were added to emulate a more realistic scenario, but the values of losses, latency and bandwidth are closely related to the communication technology adopted. Due to the random nature of the mobility model, each emulation session yields different results, so the median and standard deviation of 5 sessions are shown. Figure 5.7a illustrate the results for 5 UASs, while Figure 5.7b the results for 20 UASs for both scenarios.

Table 5.2: Emulation Scenarios

Setting	Without Losses	With Losses
Cloudlet Range	250m	250m
Number of cloudlets	4	10
Cloudlet Bitrate / Delay	433.3Mbps / 1000us	433.3Mbps / 1000us
UAS Range	90m	250m
Number of UASs	5 and 20	5 and 20
UAS Bitrate / Delay	54Mbps / 3000us	54Mbps / 3000us
Jitter	0 us	2 us
Error rate	0%	1%

The impact of the introduction of losses is perceived differently in the UAS and intra-cloudlets links. Since in this implementation, the UAS broadcasts data via UDP links (more suitable for mobile and lossy links), lost messages do not impact the broadcast latency because there are no timeouts and retransmissions. Considering *etcd* is implemented to communicate via TCP, the losses increase the overall replication latency as shown in Figure 5.7. It is important to notice in Figure 5.7b that increasing the number of aircraft while leaving the delay between broadcasts low increase significantly the end-to-end latency. This could mean that the number of replicas cannot be too high for such a platform to scale to more representative scenarios of crowded airspace. It is also important to highlight that since these were performed with an emulator, all software components related to all cloudlets and UAS, such as the clients, servers and etcd, are running in the same computer and competing for resources, and that their state can affect the results. It is, therefore, challenging to emulate scalability tests on an emulation platform since they consume too much computer resources, and more tests with a simplified replication model running on a scalable simulator are required.

5.5 Discussion, Limitation and Future Work

The emulator helped us to implement and evaluate a very first prototype of our proposal, providing quick, yet preliminary results. For instance, latency measurements with many applications running on a single computer might skew the results. The server implementation (i.e., an UDS unit) used a callback function from a *etcd* library that was not designed to have low-latency capability, especially when running multiple requests simultaneously. In this preliminary study, we also overlooked the impact of the number of replicas on the overall latency while keeping a large number of access points. We also ignored the impact of different failure models, including a Byzantine one. One important aspect of such deployments is how they will act in the presence of malicious actors, which is a reasonable assumption for an open-air platform with competing actors. We believe that the target system could tolerate attacks such as Sybil attacks [177]. Alternatively, we could consider a distributed data store based on blockchain technology to cope with Byzantine faults.

For future work, scenarios with larger areas and a different number of replicas are required. This would ensure a more precise view of the latency impact of the replication quorum. Furthermore, scalability studies are also required to simulate scenarios with hundreds of UAS. A fast simulator would also allow the study of optimal placement of such replicas in different realistic maps.

5.6 Conclusions

The different architecture modules of proposed UTM systems are still under development. We see that for safe implementation of such modules and applications, it is critical to deploy a data layer that can be used by different stakeholders with a high degree of safety and trust in data consistency because many of these stakeholders will be used as information oracles for UTM applications. This chapter introduced a tracking and position reporting system for UTM. It enables a safe and robust data sharing layer for emerging UTM services aimed at autonomous vehicles operating at very low-level airspace. Our preliminary performance evaluation suggests

that the latency measurements yielded results suitable for building a system architecture for UTM in densely populated areas. The highly available, distributed data store allows even vehicles hidden behind constructions to be spotted by all stakeholders consistently without significantly added latency. In contrast to the current ADS-B, our architecture enables fast, fault-tolerant data sharing by design and corrects some of its unsafe aspects. For future work, we plan to study how to minimize the number of replicas whilst keeping the same level of consistency and availability of the system. The UTM systems are expected to be dynamic, i.e. varying density of aircraft in different airspace areas throughout the day. Therefore, the currently available federate cloudlets could benefit the proposed architecture by allowing an adaptive fast service migration based on the current airspace state.

Chapter 6

Edge-Assisted Cost and Latency-Aware Task Offloading for a DDAA

Contents

6.1 Introduction

This chapter is the result of an on-going cooperation between ENAC and TUD[1]. Although the research is still not consolidated and published, we included in this thesis the current progress and future research we are pursuing. In this chapter we focus on scenarios related to the highest levels of UTM automation (level 4)[2], with UAS operating in BVLOS. This TCL level requires high levels of safety and automation, which require some processing running on-board with real-time constraints, and ideally not relying on centralized communication. Consequently, this reduces the communication requirements, which facilitates contingency situations. SAFE50 [77] from NASA has some assumptions regarding the UTM airspace [10], such as low altitude, high density of aircraft, urban environment, and constant V2X communication for SAA. More specifically, this chapter's main contribution is related to compute offloading applied to a DDAA system, that can be implemented on a UTM system. Sense/detect and avoid systems are characterized by two main functions: the ability to detect an obstacle on the aircraft's route that could jeopardize its physical integrity; and the ability to avoid said obstacle by creating a contingency manoeuvrer or re-route by using a path planning algorithm. Path planning algorithms in general have been extensively researched as its use is applicable to many different classes of mobile robots. There are surveys available [126, 115, 59] comparing different algorithms, such as RRT [68] and RRT* [71], potential fields [67], and Particle Swarm Optimization [6]. When dealing with constrained small factor sUAVs, it becomes challenging to run both algorithms onboard and be able to comply with real-time requirements. Therefore, we propose the alternative of using edge-assisted offloading for running high complexity tasks that otherwise would required long runtimes on constrained devices. As illustrated in Figure 6.1, the rationale is to use offloading to calculate an alternative path to deviate from a newly identified obstacle in the original trajectory.

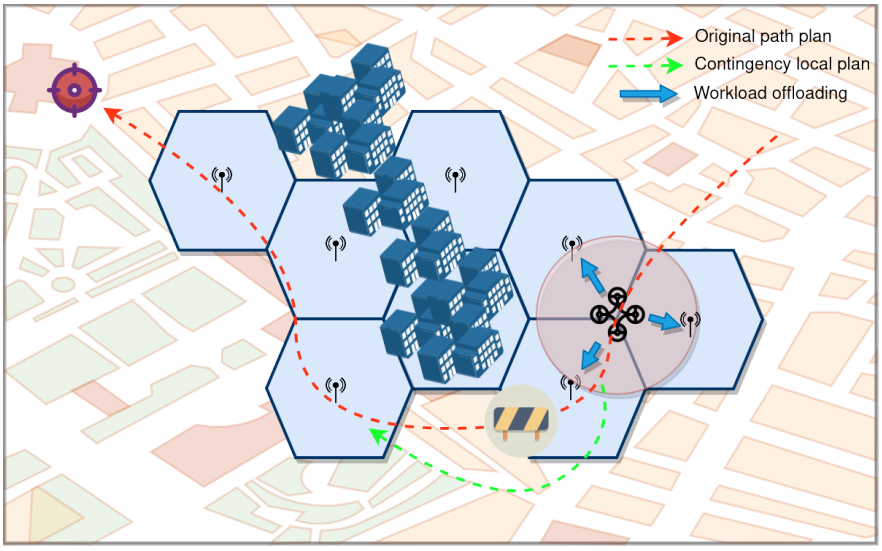


Figure 6.1: Offloading Workload for Contingency Deconfliction

[1] Technical University of Dortmund
[2] More details about the TCL levels can be found in Section 5.2.

6.1.1 Contributions

The main contributions yielded from this research are:

- Context and Motivation - We studied the current advancements and state-of-the-art in UTM systems and identified research opportunities to create a sense and avoid system for *Small Uncrewed Aerial Vehicles (sUAVs)* that can have an adequate response time to avoid significant disturbances in aircraft flight plans.
- Formalization - We formalized the problem as a partition optimization problem for one-hop vertical offloading strategies.
- Proof of concept - We created a static proof of concept using Monte Carlo algorithms as optimizers, which yielded promising results that are evidence that it is worth pursuing this view with further research. And finally, we established plans for future research to achieve all the current goals.

6.2 Context and Motivation

In this section we introduce some background knowledge related to path planning for UAS deployed on a UTM system. Chapter 5 introduced some concepts about UTM systems that will be useful in this chapter.

6.2.1 Global Path Planning

Many steps are involved in scheduling flights for UAV and sUAV. In some cases, the UAS will fly in a dense urban area with many volumetric obstacles such as buildings, towers, statues and, in some cases, natural obstacles such as hills. Therefore, it is essential as a first step to create a *global* flight plan suitable to the *world* in question. Many UAS will fly simultaneously, sharing the same airspace and sometimes the same lane in a lane-based system [137]. To create safe trajectories, it is, therefore, necessary to have in place a deconfliction system. Such safe trajectories, beyond the planned path, also include information about the time the UASs will pass each waypoint in its plan. Therefore, to be able to design these airspace-shared trajectories, it is necessary to have deconfliction algorithms. The global path planning runs on the USSs, which was introduced in Section 5.2.2.

Proactive Deconfliction

In the context of a UTM system, where many service providers share an airspace as mentioned in Section 5.2.2, they must be aware of the current state of each other. Hence, a proactive deconfliction system must have access to information about other flight plans already scheduled and to be scheduled. The deconfliction system must consider desired take-off and landing times and the origin and destination of each aircraft. With those, it can create a schedule that can accommodate all requests and guarantee that the aircraft during flight stay within a safety margin from each other, from the known obstacles and possible geo-fences[3]. One

[3]Virtual perimeter created to bound no-fly zones, or dynamically generated as a volumetric clearing around objects.

strategy to help in proactive deconfliction is to use a lane-based model[137], which schedule flights before take-off. There is, therefore, also the need for a onboard, local sense systems to detect new obstacles, and an emergency onboard path planning system to avoid them. The communication cost to perform this on a centralized system is high, so usually the focus is on decentralized systems [127].

6.2.2 Local Path Planning

Local planning is used when things are not going as expected by the global plan. The local plan must be created and followed in contingency situations, and preferably avoid drastic solutions such as returning to base [10] or performing an emergency landing. Known types of contingency are, for instance: path feasibility for contingency deconfliction, which takes place when facing unexpected ground or aerial obstacles and unsafe unforeseen atmospheric conditions, and dynamic geofencing, which can be transmitted by other aircraft or by local USS.

Contingency Deconfliction

Conflicts may also occur when a UAS goes rogue due to benign or malicious failures. Both can still be solved by a centralized UTM when the position tracking and identification systems can still be trusted. Conflicts between aircraft and a fixed or mobile unforeseen obstacle that does not participate in the UTM system cannot be solved by centralized systems due to lack of information. Onboard sensors could detect such conflicts and trigger a decentralized SAA system that would solve the problems locally by creating a local contingency plan, as illustrated in Figure 6.1. Afterwards, it would update the central system so that other flight plans can be updated to other stakeholders and that the new obstacle is also catalogued and propagated so that it can be taken into consideration by the other participant of the UTM. In [138], the authors propose a DDAA where each UAV has a model of the environment and they solve conflicts via communication exchange. In this particular proposal, a conflict can only be solved between aircraft participating non-maliciously on the UTM system, the solutions must involve alternatives using the lane-based system. In [147], the authors developed a bidding-based system, where aircraft can negotiate preferential routes during contingency situations.

6.2.3 Computation Offloading

As mentioned before, our research will focus on establishing a way that enables UASs with constrained resources to be able to run an onboard SAA system capable of offloading computation algorithms to more capable edge cloudlets. Therefore, instead of focusing on the re-routing algorithms, we focus on optimizing computing strategies compatible with the system model introduced in Section 6.4. Topologically, offloading is often applied horizontally or vertically. The former is used, for example, when the nodes offload computation to their peers. i.e. nodes with similar capabilities running in the same operational layer. The latter is used when nodes offload computation to upper operational layers with more capable devices, such as edge servers of central cloud servers. Figure 6.2 illustrates such operational computing layers, based on [142].

Offloading computation in such a dynamic environment is not trivial because it is impossible to forecast when a node will need to run the SAA algorithm. It is needed only when it detects an unexpected obstacle; hence it is not possible to pre-allocate slots in the adjacent edge cloudlets. Additionally, since the network is dynamic, it is also hard to predict how congested the network will be when offloading, which together with the bandwidth loss introduced by multi-hop communication discussed in Chapter 2, is why we decided to focus on one-hop offloading.

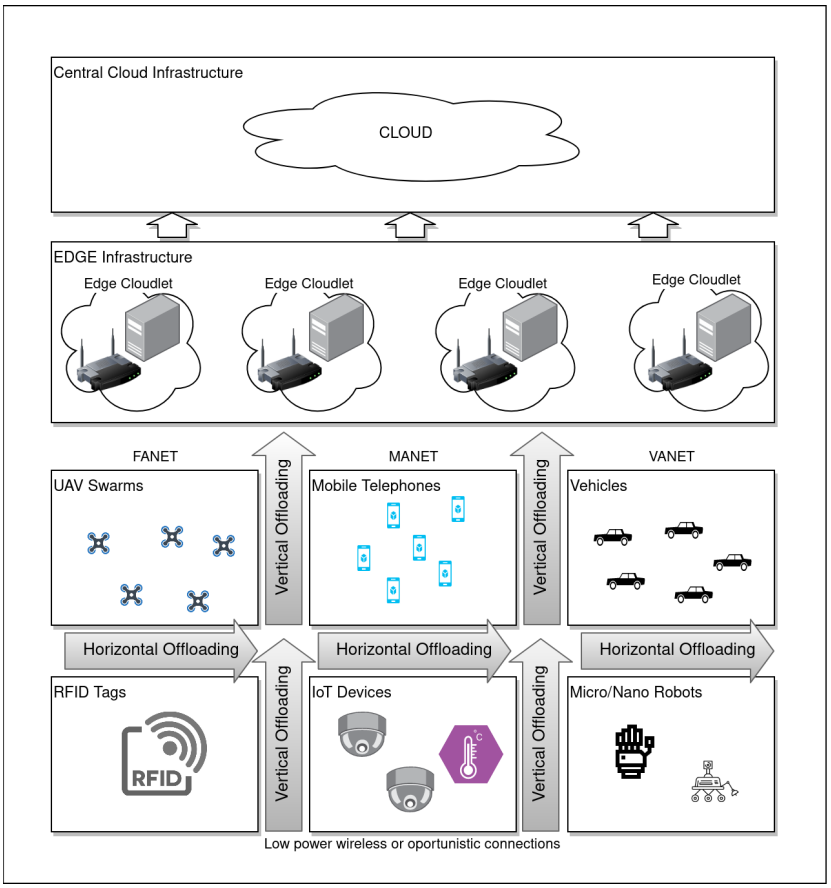


Figure 6.2: Offloading Topologies

6.3 Related Work

The horizontal and vertical offloading problem has been studied before in different scenarios. In [54], the author discusses the problem of offloading tasks vertically from a layer of IoT devices to a cloud infrastructure or edge servers. This work has a trade-off between task completion latency and energy use, which is optimised by a Lyapunov algorithm. This work requires the ability to control the CPU frequency and radio transmit power to tune the energy consumption. In [1], the system is modelled similarly, with a multi-tier offloading architecture. In that work, the offloading is modelled as a multi-object optimisation problem, where the parameters are weight according to the desired *Quality of Service (QoS)*. The

author used an accelerated particle swarm optimisation technique to solve the optimisation problem, similar but multilevel and does not take in consideration sharding the task among multiple edge servers. The authors in [122] propose a device to device offloading algorithm with focus on minimizing the energy use. Since their proposal envision an open system based on compensation, they need to consider factors that might reduce the system's reputation, such as preventing over exploitation and free riding. Allocating task has also been studied when the focus if run a batch of tasks as a scheduler, and not as a offloading problem. In [82], the author discusses a multi-UAV distributed task allocation with an auction process. Also in the same direction, the authors in [70] propose an algorithm that optimize for low latency, and to reduce SLA violations. It uses AI to control the scheduler, which is located on a central cloud server. In this paper, to create realistic scenarios, the authors mix datasets to create the scenarios, and synthetic workloads for the benchmarks. The authors in [149] also solved similar problems of optimizing task allocation vertically.

Most of the work in this area, is reduced to optimization problem [140, 139, 91, 7] applied to a specific domain [129], which means different objective function and different constraints. They usually yield high computational complexity to run, whilst some even resorting to complex AI algorithm [164] that only converge fast enough (below 100ms) when using onboard GPUs [29]. The main characteristic of our problem is that it requires a fast solution, where is acceptable not to be optimal as long as it can comply with the constraints. Monte Carlo algorithms have been used in optimization for problems that otherwise would required long computational time to converge. Due to its flexibility they can be applied to different fields [117, 181, 86], therefore it was chosen for this particular problem.

6.4 System Model

As mentioned in Section 6.1, this works focuses on the contingency path planning that is issued after take-off, i.e. during the cruise stage of the flight plan. After performing all pre-flight steps, the aircraft has a valid trajectory and schedule issued by the UTM. It then proceeds to autonomously take off and enter the cruise state, where it shall remain during the whole flight until reaching the landing site or until a contingency interrupt is triggered.

6.4.1 Contingency Model

During cruise, the aircraft control systems perform the necessary steps to follow the pre-defined trajectory. When a contingency interrupt is triggered, it needs to find a new route to deviate from the obstacle by creating a new task T_c that has a timeout $t_c \in \mathbb{R}^+$. If, before the expiration of t_c the aircraft does not converge to a feasible new route, it proceeds to emergency land, as illustrated in Figure 6.3.

6.4.2 Network Model

The network topologies, which are composed of mobile and fixed nodes equipped with wireless network interfaces, are modelled as an undirected graph $T = (N, E)$. The set of vertices N is composed of two sets $\mathcal{M}$ and $\mathcal{F}$. $\mathcal{M}$, the set of mobile nodes, which are the airborne aircraft that are part of the UTM ecosystem, composed of m nodes $\mathcal{M}_i$, with $i \in \mathbb{N}$ and $0 < i < m$.

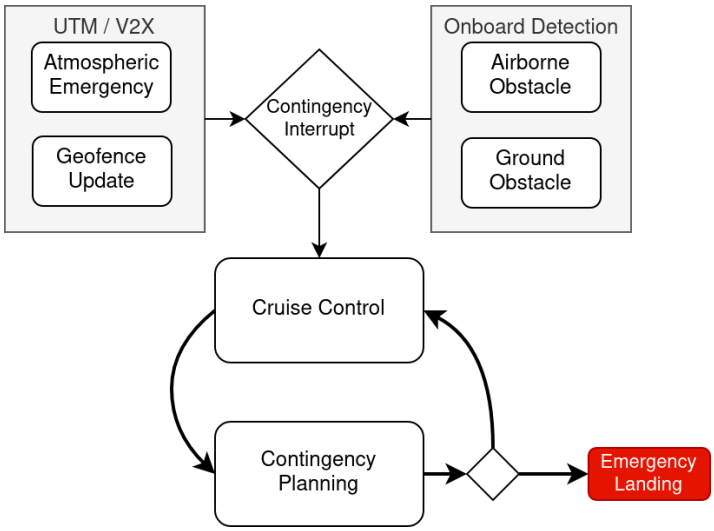


Figure 6.3: Cruise control interrupt

$\mathcal{F}$, the set of fixed nodes, which are edge cloudlets scattered over the urban area that are part of the UTM ecosystem, composed of f nodes $\mathcal{M}_i$, with $i \in \mathbb{N}$ and $0 < i < f$. The super set composed of n nodes N_i, with $i \in \mathbb{N}$ and $0 < i < n$ where $n = m + f$, is therefore $N = \mathcal{M} \cup \mathcal{M}$. The set of edges E is composed by the edges $e = (N_j, N_k)$, where the distance $d_{jk} \leq d_r$, being d_r the communication range. The range distance is defined by the wireless radio technology modelled, which allows a maximum bandwidth $B \in \mathbb{R}^+$ defined in Mbps and the network latency is considered predictable and bounded.

For simplicity, there is no membership control, and every node in the vicinity of one of the nodes participating in the connected graph T, is henceforth considered a participating node in the UTM ecosystem. For the purpose of initial implementations, the network is considered as having reliable links, so no packet loss will be taken into consideration and nodes communicate with each other when they are in direct radio range or if there is a multi-hop route between them.

6.4.3 Timing Model

In this work, it is considered that the nodes have access to a global clock. This enables processes to measure communication latencies between them and is particularly feasible when considering a swarm of UAVs since they are always equipped with one or more *Global Navigation Satellite System (GNSS)* that also supply high-accuracy timekeeping. Having a notion of synchronization and a high-accuracy global clock across the network facilitate the creation of protocols that require total ordering of messages.

6.4.4 Nodes Model

The nodes participating in the algorithm can be split into two main groups: *Edge Nodess (ENs)* and the *User Equipments (UEs)*. Nodes considered for this application are heterogeneous due to the fact that they can be composed of different CPU, SoC, SoM or *System*

on a Board (SoB) architectures and are usually equipped with different hardware solutions. Therefore, they can have processors with various architectures as long as they have a universal compute capability unit transferable among the nodes. Such devices are also composed of various amounts of volatile and non-volatile memory.

Edge Nodes (Cloudlets)

One of the design goals of this thesis is to propose a protocol capable of working within a heterogeneous environment. One way to approximate a cross-platform compute capability is to consider the number of floating point operations a processor executes per second. Even though we chose it, that also is not truly comprehensive since some architectures[4] can benefit from specific algorithms, data structures and data partitioning due to special instruction sets supplied by manufacturers, and their intrinsic architecture. For now, we will consider as a reference the processor single precision FLOPS capabilities, which usually can be found in processors' datasheets.

However, regarding communication, it will be assumed to be a homogeneous network. Each node has the same type of radio, operating the same radio medium. This way, the latency can be approximated by a well-known MAC/PHY protocol. Therefore, each edge node will have, at any moment, a state composed of the following characteristics:

- Processing capabilities in *Floating Point Operations Per Second (FLOPS)*, defined as $F \in R^+$, which is proportional to the processor capability and the current load;
- Current load capacity in Mbit, defined as $\mu \in R^+$, which is proportional to the available RAM memory;
- Current cost, defined as $c \in R^+$;
- Current bandwidth, defined $B \in R^+$, which is proportional to the distance between the EN and the UE, and the communication load;
- Position defined as $p \in \mathbb{R}^3$;

The current communication capability of the node is also essential, not only because of the technology used but also available channel usage. A node with high computational capability can help in reducing computational latency, but if its communication channels are heavily used, this might add communication latency. More details about the available bandwidth can be found in Section 6.5.4.

User Equipment (UAVs)

The offloading mobile nodes, which are also modelled as heterogeneous entities, can have different processing capacities and shall also take that into consideration when taking the decision to offload tasks to edge nodes. After the offloading data partition is defined, some remaining computation can be calculated by the UE. Each UE has at any moment a state composed of the following characteristics:

[4] GPUs, DSPs, Mainframe processors, vector processors, offloading cards such as Xeon PHI, and etc.

- Position defined as $p \in \mathbb{R}^3$;
- Velocity vector defined as $v \in \mathbb{R}^3$, containing three Cartesian components;
- Volumetric clearance radius defined as $r \in \mathbb{R}$;
- Processing capabilities in MFLOPS, defined as $\digamma_{UE} \in R^+$, which is proportional to the processor capability and the current load;

Neighbourhood

Each mobile node, while following its global path, is able to proactively create a neighbour list of one hope distant ENs. When a local contingency plan is triggered, they can eventually use it to plan an offloading strategy for the local planning to avoid the unforeseen obstacle with a certain degree of certainty. Each node periodically broadcasts *hello* packages, and each edge node in range replies with its state as defined in Section 6.4.4. Considering the topology has a set M of mobile nodes, where M_i, with $i \in \mathbb{N}$ and $i \leq |M|$, each node can have a task that is too large to be processed only by itself in the desired time constraint. Additionally, the topology also contains a set F of edge servers, where F_i, with $i \in \mathbb{N}$ and $i \leq |F|$. Each node M_i will have a list of neighbours $L_i = F_1, F_2...F_k$, where k represents the identifiers of the edge servers in direct range.

6.5 Execution Model

For simplicity, it is considered that all nodes meet all requirements for running all possible tasks. Also, all tasks can be parallelized with speed-up proportional to the number of workers (offload units) and their computing capability. When a node creates a new task to be run on the edge server, the data is split accordingly among the number of workers. They will execute the task with a proportional set to balance the load according to the design goals defined in Section 6.6. Therefore, in our scenario, the offloading is vertical, and limited to one level: from UAS to edge server.

Assumption 5. *All EN are equipped with general purpose processors that can run all offloaded tasks.*

Assumption 6. *All tasks are considered embarrassingly parallel with speed-up proportional to the amount of workers.*

Assumption 7. *The offloading node must wait all responses to be able to proceed with the algorithm, i.e. the offloading is a blocking operation.*

6.5.1 Tasks

Each task τ, can be characterized by an unique identifier i with $i \in \mathbb{N}$, the task size $\phi \in \mathbb{Z}^+$ in *Floating Point Operations (FLOP)*, the task size μ in Mbit and the task deadline t_i^d in ms.

Task Size

The task size ϕ in FLOP is estimated based on the task size μ in Mbit. Because each algorithm has an associated arithmetic complexity, which is defined by how many operations are needed per bit of data, each task will have a size in FLOP defined by $\phi = \beta\mu$. Hence, β can be parametrized to test the algorithm with different degrees of complexity. Such differentiation is important because an arithmetically intensive algorithm will benefit from a more powerful processor and suffer less with communication latency. Communication latencies hinder tasks characterized by large associated data and low arithmetic complexity.

Task Deadline

Regarding the task deadline, there is an assumption that there exist a monitoring system that can provide latency estimations with enough accuracy for our application. The reason for establishing task deadlines is to explore the feasibility of having real-time contingency local planner capabilities for a UAV with soft task deadlines. Considering such an application, the task deadline calculation can be simplified as the available time before a collision with the newly identified obstacle. A UAV which current state is defined by a position $p_n \in \mathbb{R}^3$ and its velocity $v_n \in \mathbb{R}$, and that has identified an obstacle at centre position $p_o \in \mathbb{R}^3$ and volumetric clearance radius $r_c \in \mathbb{R}$, has an avoidance time:

$$t^a = \left(\left(\sqrt{(p_{o_x} - p_{o_y})^2 + (p_{o_x} - p_{o_y})^2 + (p_{o_x} - p_{o_y})^2}\right) - r_c\right) / v_n \tag{6.1}$$

The task deadline t_i^d must therefore be established so that $t_i^d < t^a$.

6.5.2 Latency Model

When a node issues a new task that, considering its own compute capabilities, it might see as unfeasible to finish before the deadline. In such situations, it must decide if it will drop the task and perform an emergency stop or offload it to edge servers. That decision must consider its neighbourhood and optimize the task partition so that it can comply with the deadline considering the communication and processing time in each node. Hence, an objective function must be created. Currently, two goals are defined: to minimize the total turnaround time; or to reduce the offloading cost charged by the edge servers. Also, the constraints that the total turnaround time must be smaller than the task deadline and that each partition needs to comply with each server's capacity must be taken into consideration.

Offload Time

When a node offloads a task, time is spent transferring the data. The total transfer time depends on the bandwidth available between the node and the edge servers. The offloading time is defined as $t^o \in R^+$. The offloading time of task τ_i, t_i^o from node j to edge server k can be calculated with the task partition size in Mbit μ_{i_k} and the current available bandwidth between them, $\mathcal{B}_{jk}$ that are found in the neighbours list of node j, L_j:

$$t_i^o = \sum_{k}^{|L_j|} \frac{\mu_{i_k}}{\mathcal{B}_{jk}}, k \in L_j \tag{6.2}$$

Execution Time

The execution time of a task τ on an edge node N_j can be calculated by Equation 6.3.

$$t_i^e = \sum_{k}^{|L_j|} \frac{\phi_{i_k}}{\digamma_k}, k \in L_j \tag{6.3}$$

Uploading Time

After a task is finished, the executing node must upload the result to the offloading node. The upload time is defined similarly to the offloading time, but the size will be approximated by a constant μ_u.

$$t_i^u = \sum_{k}^{|L_j|} \frac{\mu_u}{\mathcal{B}_{kj}}, k \in L_j \tag{6.4}$$

The total execution time for a task is therefore $t_i = t_i^o + t_i^e + t_i^u$, and the algorithm must ensure that $t_i \leq t_i^d$.

6.5.3 Objective Function

In order to establish an objective function, the costs will be split into parts related to each objective. The computational cost is related to the renting cost charged by an edge server, since the edge servers must have an incentive to accept offloaded tasks, they charge per unit of time used. The problem of establishing costs based in competition and demand is complex and not covered by this thesis work.

Computational Cost

A task is defined by how many FLOP ϕ it requires to finish and how much memory μ it requires during the computation. The execution time of a task τ_i on a node N_j, defined by Equation 6.3, depends on the node's computing capacity $\digamma$ in FLOPS and how much of this capability is available. Since the edge server will charge a cost proportional to a fixed $\mathbb{C}^e$ (cost per unit of time) during the task execution t_i^e, the total task execution cost on node N can be calculated by the Formula 6.5. The costs, as referred here, can be defined as abstract or actual monetary renting costs if the edge servers are federated.

$$C_i^N(\phi, \mu, t^d, L) = t_i^e \mathbb{C}_N^e \tag{6.5}$$

Latency Cost

Considering the time to offload a task i, t_i^o the execution time t_i^e and the time to upload the results t_i^u as the total latency t_i^N for running the task on node N. We can calculate the latency cost with Formula 6.6. The cost here is defined as an abstract cost, and total latency is henceforth referred to as turnaround time.

$$\mathcal{L}_i^N(\phi, \mu, t^d, L) = t_i^N \mathbb{C}_N^l \tag{6.6}$$

Cost Function

An *overall cost function* can therefore be defined as Equation 6.7, and every time a node wants to offload tasks to its neighbours, it must minimize the former.

$$f_{C_\tau}(\phi, \mu, t^d, L) = C_{e_N}\alpha + C_{l_N}\gamma \tag{6.7}$$

Therefore, when needed, a middleware configured with two parameters α and β to define the priority goals and will receive as an entry for each task $\tau_i = \{\phi, \mu, t^d, L\}$:

- ϕ_i - The task number of FLOP
- μ_i - The task memory size in B;
- t_i^d - The task deadline in ms
- L - The list of neighbours

The middleware will produce as an output an array with the same length as the list of neighbours where each element is a percentage of the task total size $\omega_n \in R^+$ where $0 \leq \omega_n \leq 1$ and $\sum_0^N \omega_n = 1$, as shown in Figure 6.4. The offloading node can split the task and distribute it accordingly. In some cases, depending on the costs, it could happen that part of the task would be processed locally, avoiding the offloading latency. Therefore, the objective is to generate an offloading partition $P = [\mu_0, \mu_1, \ldots, \mu_{|L|-1}]$, where $\mu_n = \omega_n \mu$, subject to the offloading cost being minimized without violating the deadline.

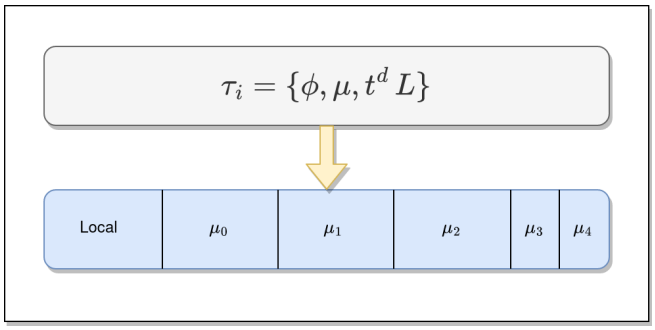


Figure 6.4: Task Split Illustration with four Edge Cloudlets.

Initially, we will treat the problem split into two distinct objectives, minimize the offloading cost or the turnaround time. The offloading cost is the price to be paid to perform computation on an edge server. The price charged by the edge servers is usually variable and modelled as a distribution function. We will, for simplicity, define a fixed cost. As there might be an unpredictable number of obstacles triggering offloading actions along the path of a UAV, we focus on minimizing the cost each time the offloading function is triggered. The objective function for a task τ_i can be modelled as follows:

$$\begin{aligned} \text{minimize} \quad & C_i^N(\tau_i), \\ \text{subject to} \quad & \sum_{j=0}^{|L|-1} \mu_j + \mu_{local} = \mu \\ & \forall j \in L, \mu_j \leq M_j, \\ & T(P) \leq t^d \end{aligned} \tag{6.8}$$

or, when minimizing the turnaround time:

$$\begin{aligned} \text{minimize} \quad & \mathcal{L}_i^N(\tau_i), \\ \text{subject to} \quad & \sum_{j=0}^{|L|-1} \mu_j + \mu_{local} = \mu \\ & \forall j \in L, \mu_j \leq M_j, \\ & T(P) \leq t^d \end{aligned} \tag{6.9}$$

where $\mu_j \geq 0$ is the memory size of the workload to be offloaded to edge server j, M_j is the amount of available memory on edge server j, f_{L_j} is the cost function of edge server L_j, and T is the turnaround time of task τ under the offloading plan P.

6.5.4 Parallel and Sequential Offloading

As mentioned in Section 6.4.2, the offloading nodes (UAVs) and cloudlets are equipped with wireless interfaces. Hence, how data is transferred during the offloading can have a considerable impact on the turnaround time and system performance. In a real-world deployment, contention, RF interference and lack of reliability of the links need to be taken into consideration. When identifying the offloading strategy, we will divide the data transfer into two macro-categories: parallel and sequential offloading.

While the bandwidth provided by wireless technology is usually fully available for a node, considering that all connections might share the same channel, this system's bandwidth is shared among all the nodes exchanging data with offloading node. Considering the set of neighbouring nodes $L(t)$ being candidates for node j offloading cloudlets, the bandwidth available for the link between nodes j and each cloudlet as seen at node j is:

$$B_{jk}(t) = \frac{B}{|L(t)|} \tag{6.10}$$

However, the receiving node k will eventually serve other offloading nodes. So, the actual bandwidth available $\tilde{B_{jk}}$ might be lower than what is expected; i.e., every link will be limited by the lowest available bandwidth between two nodes. Considering any destinations n being served by k:

$$B_{jk}(t) = \begin{cases} B_{jk}(t), & \text{if } B_{kn}(t) > B_{jk}(t) \\ B_{kn}(t), & \text{otherwise} \end{cases} \tag{6.11}$$

To evaluate the behaviour of the data transfer when offloading a task partition, we made

some tests under realistic network conditions using the MACE [46] emulator. We used as a test platform the computer described in Table 6.1, using EMANE to simulate the MAC/PHY configured for IEEE 802.11g (54Mbps). The network topology is presented in Figure 6.5, where the node with ID 0 is a offloading UAV, and the other nodes are edge cloudlets. In this test, an application runs in each node and creates a list of neighbours using UDP *hello* packets. The UAV node copies binary data via a TCP socket to the cloudlet nodes. It records how long it took to transfer the files, how long it waited when offloading serially, and estimates how long it took to process the task based on the computing capacity and the arithmetic complexity as stated in Table 6.2.

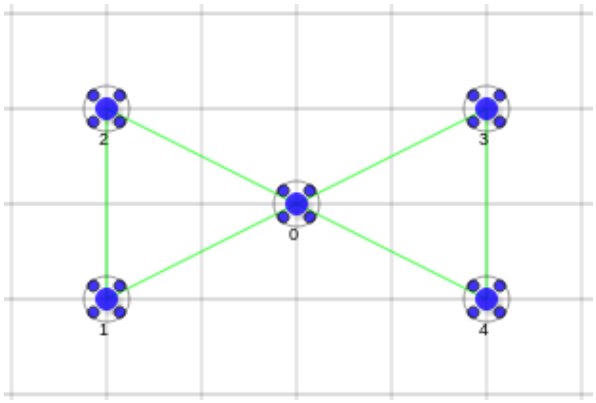


Figure 6.5: Example of Parallel Offloading.

Table 6.1: Test Platform.

Processor Manufacturer / Model:	Intel / i7-8550U
Number of cores	4 (8 Threads)
Memory	16GB DDR4
Emulator	CORE 8.2.0 and Emane 1.3.3
Operating System	Linux Mint 21 x86 64bits
Routing Protocol	B.A.T.M.A.N IV, when needed

Table 6.2: Scenario Configuration

MAC:	IEEE 802.11g
Task Sizes	100KiB, 250KiB, 500KiB, 600KiB
Task Complexity (β)	2
Cloudlet F (MFLOPS)	4.5

For simplicity, we do not take into consideration the result upload time since it is negligible compared to the other times.

Parallel Offloading

In parallel offloading, the task-generating node will have the shared bandwidth available for each task partition as determined by Equation 6.11. In this situation, there is no waiting time

for the partitions, and as soon as the cloudlet receives the data, it can start computing it. An example illustrating a situation with four available cloudlets receiving partitions of different sizes is shown in Figure 6.6. Since the tasks are sent in parallel, as soon as the first smaller task finishes transferring, it starts running and finishes before the others. Additionally, as soon as each task finishes transferring, the other tasks have more available bandwidth and accelerate the transfer. It is noticeable that each task will finish at different times, which is not interesting for some applications such as the DDAA where the UAV needs to wait for the result of all tasks before deciding for this particular example after 600ms.

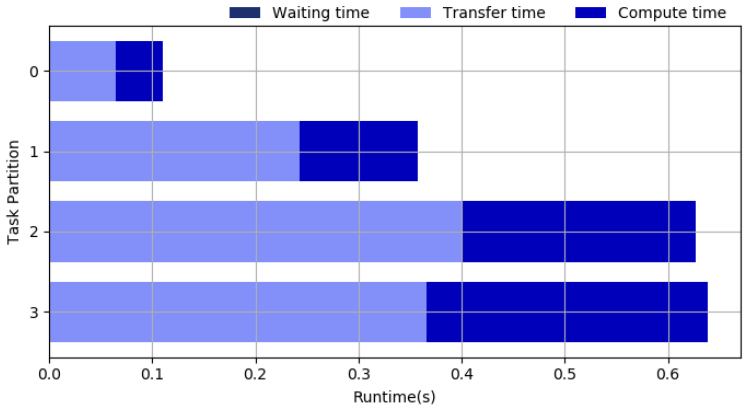
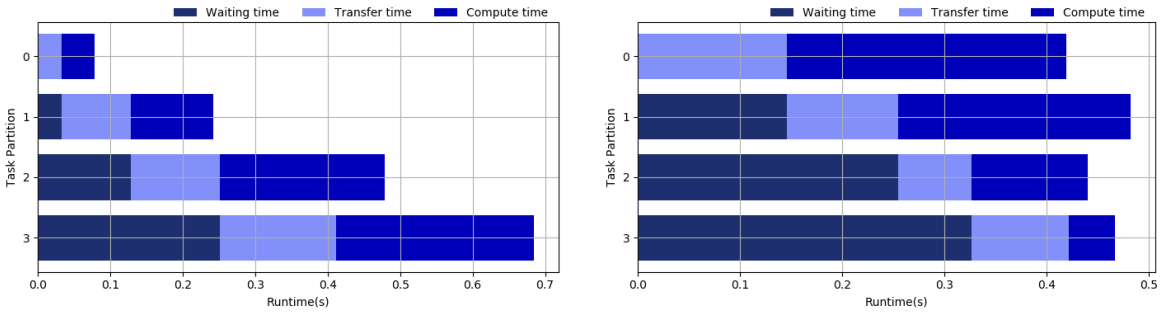


Figure 6.6: Example of Parallel Offloading.

Sequential Offloading

Sequential offloading can be defined in two distinct ways regarding the order of the task. Send the tasks from the larger to the smaller or the other way around. The main difference is that contention and bandwidth sharing are avoided when sending sequentially. Hence, the sending node always has the entire bandwidth for each task, accelerating each upload. Also, in this scenario, each task's waiting time added to the overall latency. The results are shown in Figure 6.7.



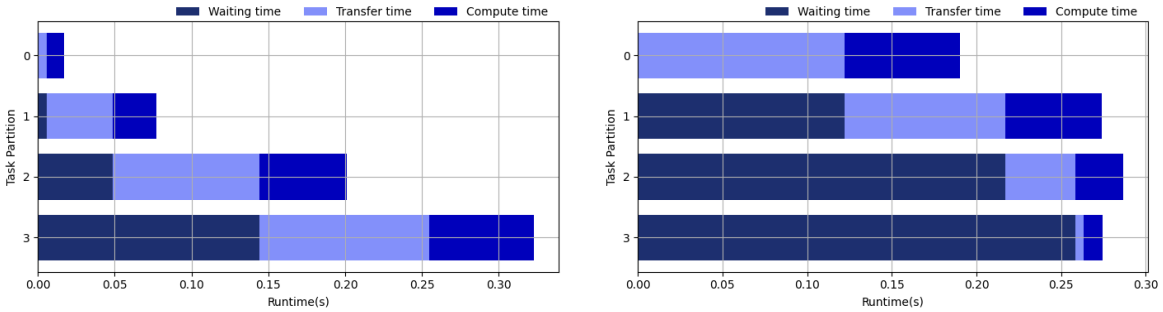
(a) From Smaller Tasks to Larger Tasks.

(b) From Larger Tasks to Smaller Tasks.

Figure 6.7: Sequential Offloading

One positive effect is observed in this scenario when sending tasks from the largest to the

smallest, as shown in Figure 6.7b. Since the tasks take longer to run than to transfer, even though there is an added waiting time, the tasks finish closer to each other when compared to the other case shown in Figure 6.7a. Also, the overall time is smaller, less than 500ms. It is essential to highlight that this will not always be the case since it depends on the task complexity β and the cloudlet capacity $\digamma$. Figure 6.8 shows the same tests, but with a cloudlet with more compute power available $\digamma = 9$ FLOPS, and more simple tasks with $\beta = 1$.



(a) From Smaller Tasks to Larger Tasks.

(b) From Larger Tasks to Smaller Tasks.

Figure 6.8: Sequential Offloading with More Computing Power.

In this case, it is possible to see that the computing time lowers while transfer time remains the same, which does not change the fact that the sequential offloading in this order is better. Therefore, we will adopt the larger-to-smaller order sequential transfer since complex tasks can only benefit from it, and simpler tasks do not harm the performance.

6.6 Offloading Strategies

We initially considered two different strategies for offloading tasks. Cost-based offloading, based on the offloading cost charged by the edge servers. Turnaround-based offloading, based on the turnaround time of an offloading task. To enable the strategies, we must solve an optimisation problem. Using brute force to solve such problems yields an NP-hard problem, which cannot be solved in polynomial time. As mentioned in Section 6.3, such problems have been studied before and have solutions that avoid brute forcing to converge faster. However, still not fast enough for a UAV to avoid a collision during cruise velocity, when considering a task partition offloading problem. We decided to use Monte Carlo optimisation strategies to minimise the problems in Equations 6.8 and 6.9.

6.6.1 Optimization

Monte Carlo algorithms are often used to solve NP-Hard problems due to the possibility of reaching a satisfactory solution in a considerably low runtime [117, 181, 86]. Although the solution might not be optimal, if it is close enough to solve the collision avoidance problem before the task deadline, it is considered a satisfactory solution. The rationale behind these algorithms is to use aleatory values to calculate new solutions, and each algorithm has a different way of doing so and escaping local minima. The scaffolding solution that will be

used is presented as the pseudo Algorithm 4. In the upcoming sections, we will show how to adapt this solution using different techniques.

Algorithm 4 Monte Carlo Scaffold.

```
Ensure: t^t < t^d                          ▷ Turn around time smaller than task deadline
  i ← 0                                    ▷ Global iteration
  k ← 0                                    ▷ Local iteration
  P ← new_partition()                      ▷ Initial partition
  P_c ← cost(P)                            ▷ Initial partition cost
  while k < k_max do
    P* ← new_partition()                   ▷ New partition candidate
    P*_c ← cost(P*)
    if P* is valid then
      Δ_c = P*_c − P_c
      if (Δ_c < 0) then
        P ← P*
        P_c ← P*_c
        k ← 0                              ▷ Reset Local iteration
      end if
    else
      continue
    end if
    k+ = 1                                 ▷ Increment Local iteration
    i+ = 1                                 ▷ Increment Global iteration
  end while
```

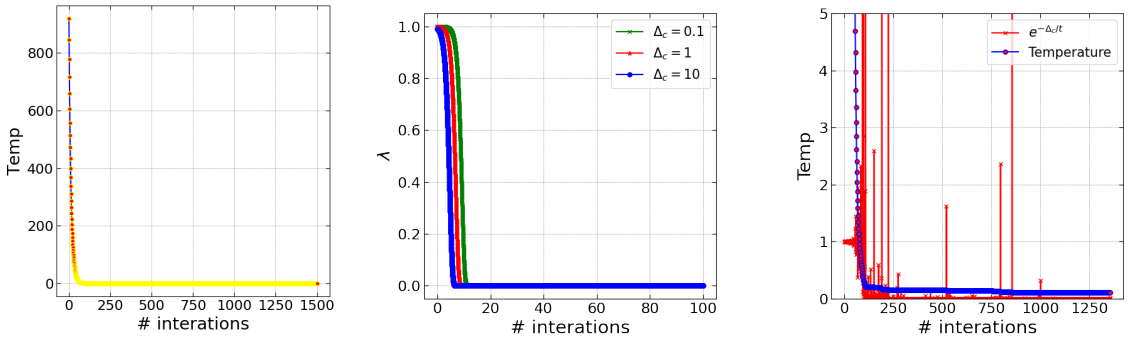
Here the function *cost* refers to cost, turnaround time, or overall cost depending on the strategy adopted. The local maximum iteration k_{max} ensures that the program stops after a maximum number of iterations. The solution space has the same dimensions as $|L|$, so at each iteration, when creating a new partition candidate P^*, only a pair of edge servers will be randomly selected to change their partition sizes. Consequently, what is removed from one, is added to the other. When a new partition yields a lower cost, this partition is selected as the current best. The longer it is allowed to run, the better the solution will be, but the best partition might not be optimal or be stuck at a local minimum.

6.6.2 Simulated Annealing

Among the Monte Carlo solutions, the *Simulated Annealing* is often used as an optimization algorithm. It mimics the annealing process used in metallurgy, where a metal league is heated and cooled down so that the molecules can accommodate and form a more resistant material. As the Monte Carlo scaffold, the algorithm uses random sampling for the task partition problem to create a new partition candidate P^* at each iteration. When a better partition is found, it is adopted as the current best. If a maximum number of iterations k_{max} passes without a new best solution being found, the current best is considered the close to optimal partition P. An important aspect of the simulated annealing, is the abstraction of a temperature cool down. Although there is no direct relation between the abstract temperature and the optimized objective, While the former is high, it also accepts a solution with higher cost. This higher value might be accepted according to the value of $\lambda = e^{-\Delta_c/t}$, and it is

accepted with a certain probability that declines as the temperature gets colder. Hence, it depends on the current temperature t and Δ_c's value, which is the cost difference between the best and the candidate partitions. Temporarily accepting worse solutions with a decaying probability allows for exploring better the solution space and escaping local minima.

Figure 6.9a illustrates the temperature decay, while in Figure 6.9b λ. Looking closer at lambda and the temperature, as shown in Figure 6.9c, it is possible to see the effect the temperature has on λ, and the probability of choosing a worse value for the objective function. For every iteration, if lambda is greater than a random number r with $r \in \mathbb{R}^+$ and $0 \leq r < 1$, it will accept a worse solution. Hence, as shown in Figure 6.9c, the colder it gets, the smaller the probability that this will happen.



(a) Temperature decay. (b) Lambda variation. (c) Lambda variation.

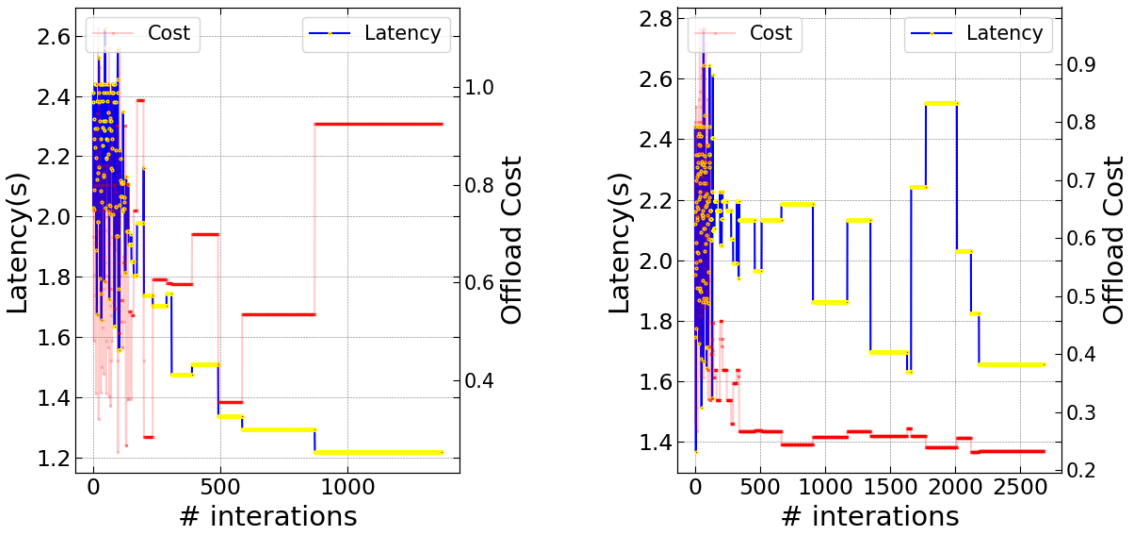
Figure 6.9: Simulated Annealing.

The simplified algorithm for the simulated annealing optimization is shown in the pseudo Algorithm 5.

Algorithm 5 Simulated Annealing.

Ensure: $t^t < t^d$ ▷ Turn around time smaller than task deadline
  $i \leftarrow 0$ ▷ Global iteration
  $k \leftarrow 0$ ▷ Local iteration
  $t \leftarrow t[0]$ ▷ Initial temperature
  $P \leftarrow new\_partition()$ ▷ Initial partition
  $P_c \leftarrow cost(P)$ ▷ Initial partition cost
  while $k < k_{max}$ **do**
    $P^* \leftarrow new\_partition()$ ▷ New partition candidate
    $P_c^* \leftarrow cost(P^*)$
    if P^* is valid **then**
      $\Delta_c = P_c^* - P_c$
      if $(\Delta_c < 0)$ OR $(e^{-\Delta_c/t[i]} > random\_number())$ **then**
        $P \leftarrow P^*$
        $P_c \leftarrow P_c^*$
        $k \leftarrow 0$ ▷ Reset Local iteration
      end if
    else
      continue
    end if
    $k+ = 1$ ▷ Increment Local iteration
    $i+ = 1$ ▷ Increment Global iteration
  end while

Figure 6.10 illustrates the behaviours of latency and renting costs during the optimization process when optimizing for different objectives. The total number of iterations is different since it is configured to stop when a certain number of iterations pass without significant change in the optimized solution. Moreover, as expected, when optimizing for one objective, the other is allowed to run free and not necessarily achieve optimal values.



(a) Latency optimization.

(b) Cost optimization.

Figure 6.10: Cost Variation with Simulated Annealing Optimization.

6.6.3 Genetic Sampling

We introduce a simple implementation of the genetic algorithm applied to our offloading application. There are three main elements in genetic optimization, which are shown in Figure 6.11. The gene is the value that will be explored in the solution space. In this application, it represents the share of the task offloaded to a server. The chromosome is the collection of genes or a candidate solution. And the generation is the population containing the current mutated chromosomes. The rationale is intuitive: when a server has low computing capability, which affects the turnaround latency, or a prohibitive renting cost, which affects the overall cost, its gene will naturally be omitted and selected out after some generations.

So, at each generation, a new mutation in one chromosome is induced. It changes how much will be offloaded to that server, consequently adding the difference to another server. When all mutations are created, equal to the number of genes, we calculate which one is the fittest, i.e. which yields the lowest cost according to the objective. The fittest chromosome will be selected as the base for the next generation. The process continues until a maximum number of generations happen or when the objective cost is close enough to a predefined threshold. The simplified algorithm for the genetic sampling optimization is shown in the pseudo Algorithm 6, while for the mutations in presented in the pseudo Algorithm 7.

Algorithm 6 Genetic Sampling.

Ensure: $t^t < t^d$ ▷ Turn around time smaller than task deadline
  $i \leftarrow 0$ ▷ Global iteration
  $k \leftarrow 0$ ▷ Local iteration
  $t \leftarrow t[0]$ ▷ Initial temperature
  $P \leftarrow new\_partition()$ ▷ Initial partition
  $P_c \leftarrow cost(P)$ ▷ Initial partition cost
  while $k < k_{max}$ **do**
    $P^*[n] \leftarrow fork_n(mutation())$ ▷ New n mutation candidates
    $P_c^* \leftarrow mincost(P_[^*m])$ ▷ Get best candidate
    $\Delta_c = P_c^* - P_c$
    if $(\Delta_c < 0)$ **then**
      $P \leftarrow P^*$
      $P_c \leftarrow P_c^*$
      $k \leftarrow 0$ ▷ Reset Local iteration
    end if
    $k+ = 1$ ▷ Increment Local iteration
    $i+ = 1$ ▷ Increment Global iteration
  end while

Algorithm 7 Chromosome Mutation.

  $k \leftarrow 0$ ▷ Local iteration
  while $k < k_{max}$ **do**
    $P^* \leftarrow new\_partition(id)$ ▷ New partition candidate
    if $is_valid(P^*)$ **then**
      $return(P^*)$
    else
      $continue$
    end if
    $k+ = 1$ ▷ Increment Local iteration
  end while

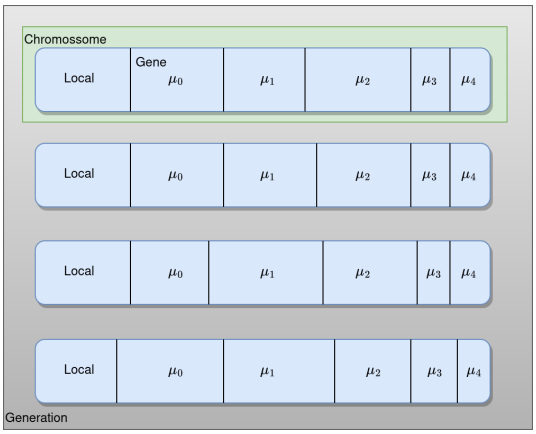


Figure 6.11: Illustration of Genetic Algorithm Components when Applied to Task Offloading.

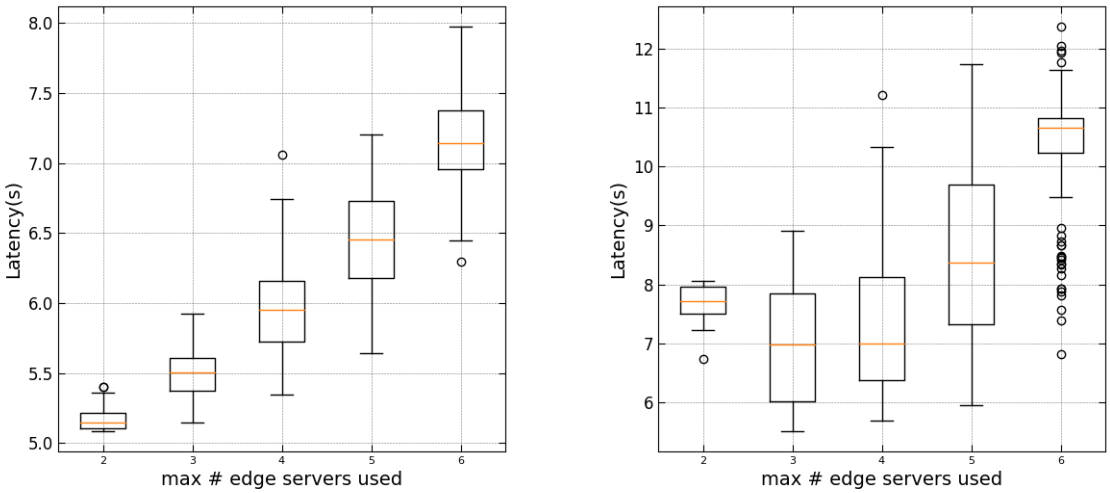
6.7 Offloading Strategy Evaluation

As a first attempt to evaluate the optimization strategies described in 6.6.1, we implemented the algorithms and optimizers in Python and ran the scenario described in Table 6.3.

Table 6.3: Simulation Scenario.

Server	1	2	3	4	5	6
Cost	3	6	7	1	2	6
Speed (MFLOPS)	90	60	60	80	90	90
Bandwidth (Mpbs)	22	24	21	21	25	26
Capacity (Bytes)	40	20	10	70	60	20

The scenario is static but can be easily converted to a dynamic simulator converting the values in Table 6.3 to values derived from probability distributions. It is also configured to have a maximum local calculation of 10% of the task. First, we show the implementation of the simple Monte Carlo scaffolding Algorithm 4. In this implementation, we can manually impose the maximum number of edge servers allowed to be used. Figure 6.12a shows what happened to the final latency when optimizing for the lowest turnaround time, with an increasing number of the available edge servers used. It shows evidence that using the least servers yields better results, which should also be observed when using other optimization algorithms without manually limiting the maximum number of servers. Figure 6.12b shows the total latency, but now when optimizing for renting cost. As expected, the latency values are less optimal than when optimizing for turnaround time.

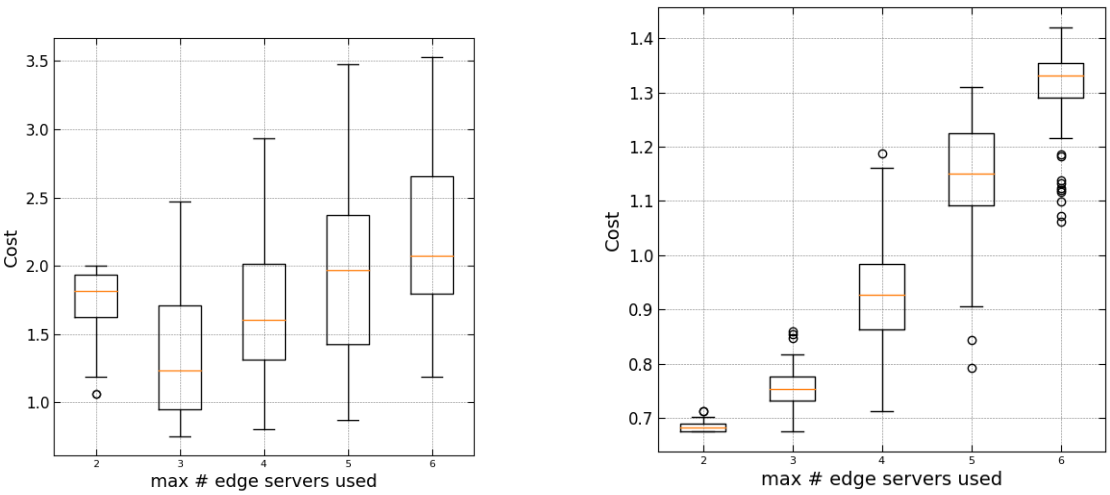


(a) Optimized for Turnaround Time.

(b) Optimized for Renting Cost.

Figure 6.12: Final Latency Values for pure Monte Carlo Optimization with Different Number of Edge Servers

In the analogous way, Figures 6.13a and 6.13b illustrate the behaviour of the renting cost when optimizing the partitions for turnaround time and renting cost, respectively.



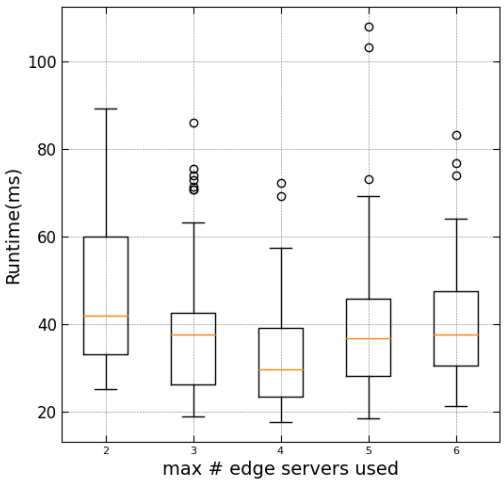
(a) Optimized for Turnaround Time.

(b) Optimized for Renting Cost.

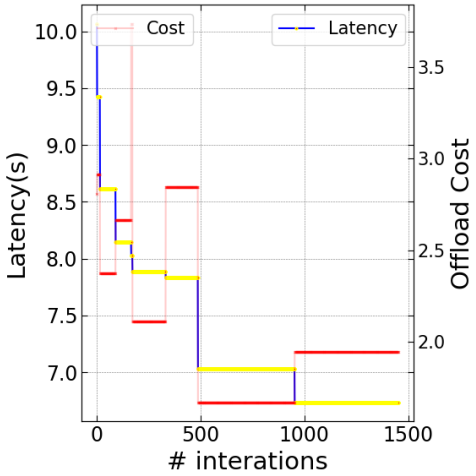
Figure 6.13: Final Renting Cost Values for pure Monte Carlo Optimization with Different Number of Edge Servers

One aspect that is extremely important in this application is the algorithm runtime, i.e. the time it takes to optimize the partition before being able actually to offload the task.

Since we aim at calculating, offloading and evaluating the results in sufficient time to change the route and avoid the collision, it is ultimately important that the optimization takes a manageable amount of time. Figure 6.14a shows the runtime for the pure Monte Carlo optimization, which runs, on average, in less than 50ms for this scenario.



(a) Total Runtime for Running the Optimization.

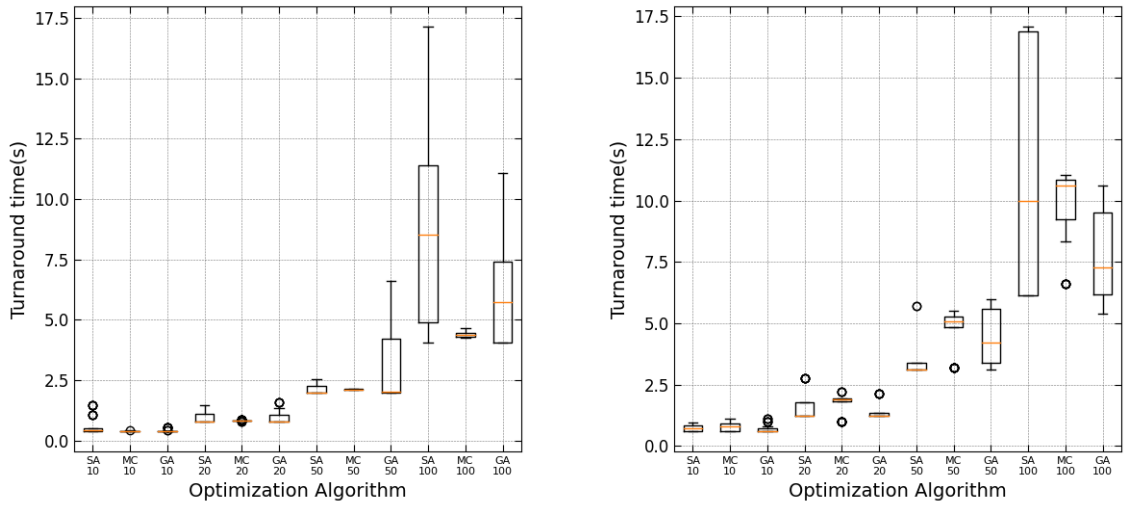


(b) Cost evolution for 6 Servers Optimizing for Turnaround Time.

Figure 6.14: Final Runtime Values for pure Monte Carlo Optimization with Different Number of Edge Servers

We now show the results in Figures 6.15 and 6.16 for the three implementations with increasing task size to offload after running each simulation 50 times. We now configure the pure Monte Carlo to use a maximum of two edge servers and let Simulated Annealing and Genetic implementations naturally select how many edge servers they will use. Figure 6.15a shows latency results when optimizing for the lowest turnaround time. All three implementations achieve similar results with smaller task sizes, hinting that this might be close to an optimal solution. However, the current implementation of the simulated annealing and genetic algorithms yields higher values with larger tasks. This might indicate that the implementation needs fine-tuning, such as higher temperature, to escape local minima. As expected, the results for latency when optimizing for lower renting cost are significantly worse, as shown in Figure 6.15b.

The results for renting cost are shown in Figures 6.16a and 6.16b. The results are similar, but the simulated annealing performs better, which was not the case when optimizing for turnaround time. As mentioned earlier, low runtime is very important for this particular application, and the current implementation of the genetic algorithm is experiencing higher runtime when compared to the other algorithms, as shown in Figure 6.17. This is the case because the generations are created in parallel and compared after the threads are joined. If one of the workers takes longer to find a feasible solution, the main thread needs to wait for it to be able to compare all solutions.

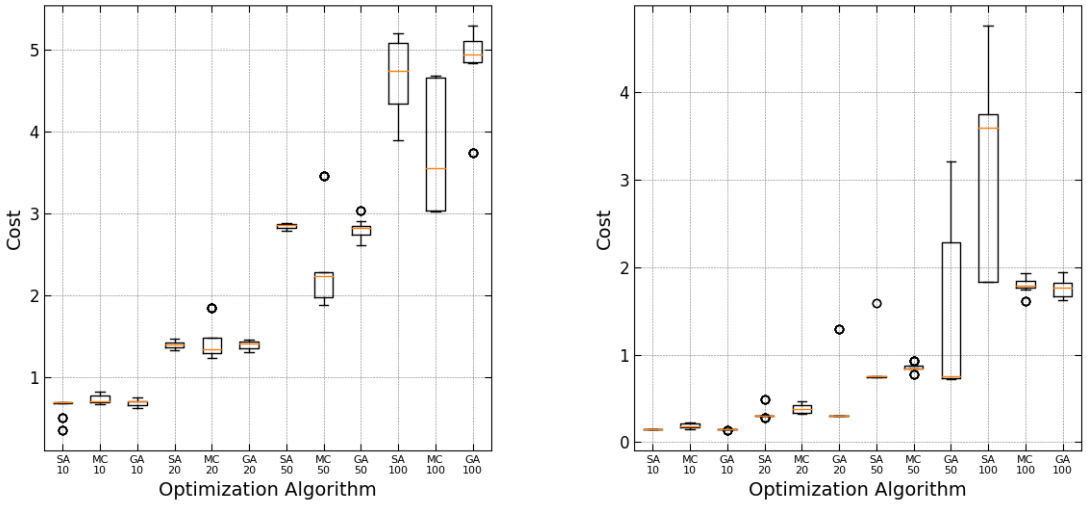


(a) Optimized for Turnaround Time.

(b) Optimized for Cost.

Figure 6.15: Comparison of Turnaround Time with Increasing Task Size and Different Optimization Algorithms.

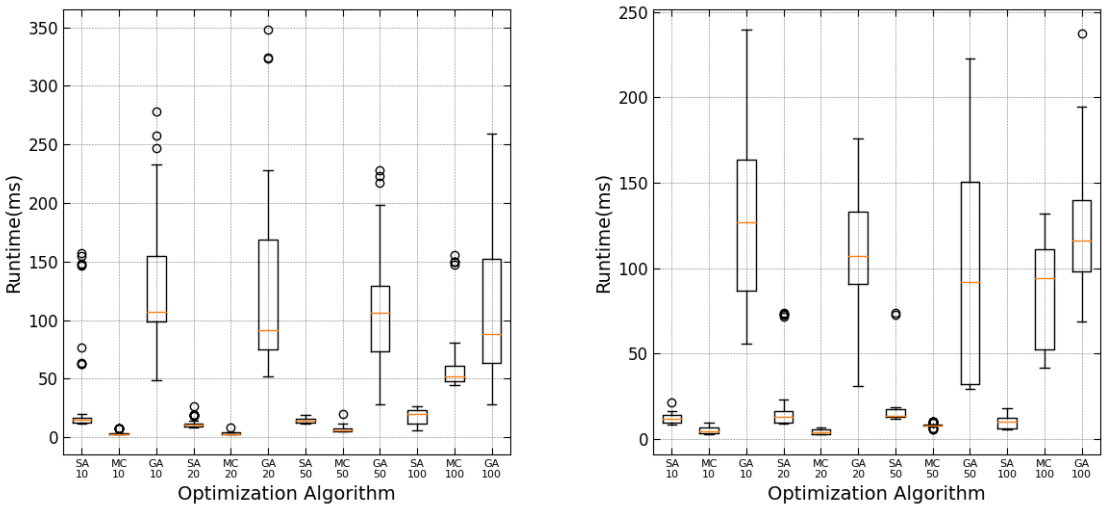
One interesting result is that when optimizing for more complex solutions (larger tasks), the simulated annealing runs significantly faster than the pure Monte Carlo implementation.



(a) Optimized for Turnaround Time.

(b) Optimized for Cost.

Figure 6.16: Comparison of Renting Cost with Increasing Task Size and Different Optimization Algorithms.



(a) Runtime Optimized for Turnaround Time.

(b) Runtime Optimized for Cost.

Figure 6.17: Comparison of Runtime with Increasing Task Size and Different Optimization Algorithms.

6.8 Discussion and Future Work

In this chapter we showed that the proposed strategy, implemented as a proof of concept simulator, has revealed interesting results. It showed that it is worth exploring further with more detailed and variable scenarios. There is, however, future work to enhance the implementation of the Genetic Algorithm, especially in the direction of potentially disabling problematic genes, which would avoid waiting times and enhance the performance.

The subject of this chapter has been studied in a cooperation project together with the University of Dortmund, and since it is still an ongoing project, there are some outstanding implementations. The main future directions now are to implement the application in the simulator introduced in Chapter 3 and in the Robot simulator designed by the authors in [123]. To evaluate the proposed strategies in a more realistic scenario, we propose the use of a dataset [63] that contains the location of edge towers distributed in the city of Shanghai to be used as a reference for possible offloading stations. It is expected that with the advancement of 5G, the number of edge cloudlets available will increase, making this application feasible for industries without significant investment. A square area with approximated 5km sides, illustrated in Figure 6.18, was chosen as the place where UAVs will be deployed with predefined flight paths. Several randomly generated routes were created using RRT^* to avoid the buildings. The map and buildings were downloaded from OpenStreetMaps's API using the OSMnx Python library [113], which also allows the introduction of extra geo-fences if needed.

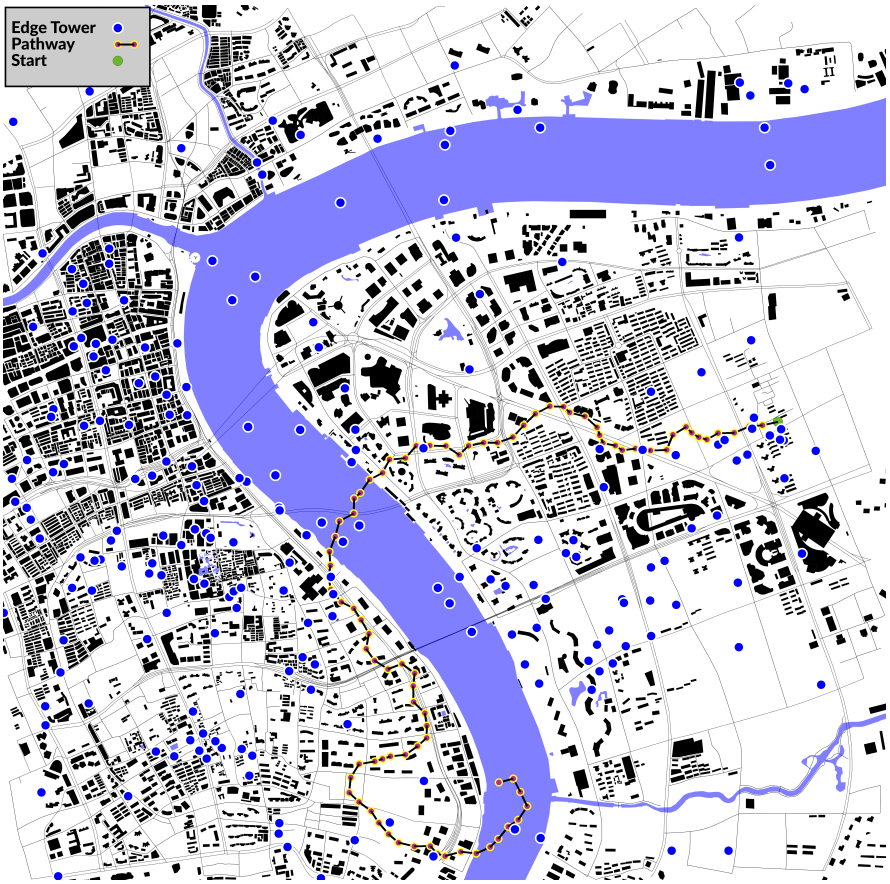


Figure 6.18: Map of the Shanghai chosen for simulations. Area limits: 31.25684°N; 31.21332°S; 121.52716°E; 121.47574°W. Includes representation of edge towers and one example global path.

The simulation, therefore, runs by making the UAV follow these waypoints while randomly generated obstacles are placed on it's way, which is an option already implemented in the simulator. When a UAV forecasts a collision with such obstacles, it offloads a workload to nearby edge cloudlets in an attempt to generate a contingency local plan to avoid the obstacle without stopping or reducing the velocity. The work with the simulator is advanced, but require more time for testing and evaluation.

Chapter 7

Conclusions and Future Work

Research in distributed systems deployed in dynamic networks has gathered much attention recently. Even though distributed algorithms and cloud-based services have been extensively studied, such time-varying topologies introduce new challenges, especially when considering specific desired properties for the correct behaviour of distributed systems. The challenges span from the physical communication medium, the physical environment where the nodes are deployed, to the intrinsic constraints of devices commonly used in Cyber-Physical Systems. Cyber-Physical Systems, as described in Section 2.2, are a class of devices that combine computers (cyber) and physical actuators, to perform critical tasks. When several of them are combined interactively, and they cooperate in solving a task, we name them as a swarm of CPS. Such swarms, usually connected via MANETs, are crucial for some missions since they increase their robustness even though they are deployed in dynamic networks. When designed with high levels of redundancy, one failing node does not necessarily jeopardize the whole mission. And, when built with appropriate design goals, they can be deployed in hard-to-access, distant and dangerous places where human access would otherwise be unfeasible. Often, these swarms run distributed algorithms that require balanced and controlled network traffic to function with the designed performance. In Section 2.2.3, we discussed how the CAP theorem relates to such applications in swarms of CPS, showing that it is crucial not only for the correct application function, but also for the swarm control itself. Nodes' mobility and failure-prone network communication increase the probability of creating network partitions. It also affects the availability and consistency of distributed applications deployed in such topologies as described by the CAP theorem.

The impact of network traffic on swarm intelligence and distributed applications was shown via experimentation in Section 2.2.7, which is why we focused on the challenges imposed by communication constraints in this thesis. Furthermore, we showed, via experimentation in Section 2.3.2, the impact that network density has on a MANET's connectivity and its ability to exchange information, confirming the existing theories' claims. That was one of the main reasons we kept the goal of using existing software as an initial effort for this doctoral project. Furthermore, we leverage them to create appropriate methodology to simulate, to observe and to instrument distributed applications running with good mobility models and communicating via failure-prone wireless networks. Therefore, it was valuable to study the behaviour of flows in the network running on various topologies, scenarios and applications. That induced a major doctoral project's goal: to establish complementary methodologies for instrumenting and analysing qualitative data in CPS swarms via emulation and simulation. By developing such methodologies, we showed experimentally that other characteristics, such as network connectivity in Section 2.3.1, are measurable and that they undermine the nodes' ability to function as a swarm and the applications' performance. This was why so much effort was put into having proper emulation and simulation as methodology cornerstones. We exposed

during the thesis the rationale behind the development of these tools and their limitations.

7.1 Contributions

The main contribution of this thesis was to create methodological tools that can support future scientific projects. And, our emulator has already been used by laboratory colleagues to support and publish their research. Furthermore, we demonstrated how these methodologies can be used to develop protocols and algorithm that have a direct impact on our society. A lightweight fluid model was introduced in Chapter 3, implemented, validated and tested. Using fluid models to simulate message passing as continuous flow instead of a discrete simulator, provide a high level of scalability, which allows the simulation of hundreds of nodes with high level of network traffic. We also provided additional models to it so that it could be used as a fully functional simulator for CPS swarms, whose usage was demonstrated through different data flow models, mobility models and topologies. Indeed, it constitutes a versatile tool since different distributed services can be modelled on it. Additionally, due to its modularity, it is straightforward to introduce new measurements and functionalities, as we showed by adding realistic mobility control models to study swarm intelligence.

The MACE emulation framework, introduced in Chapter 4, aimed at testing and prototyping applications, with a focus on CPS swarms and mobile computing in general, was built on top of existing state-of-the-art network emulators. We introduced new features needed for fast distributed services prototyping, instrumentation and emulated deployment that were not present in previous work. This tool was crucial for this doctoral project and was used in most case studies, leaving its footprint on all chapters of this thesis. It was used to validate proposed models, to develop UTM related prototypes, and even to enable such prototypes to use off-the-shelf software such as *etcd*. Experiments run on it were easily reproducible with the aid of automated scripts. Even though it is impacted by the emulation side effects, such as execution in wall-time and competition for the computer resources by the several components, it was crucial to enable this thesis.

After building these two methodological tools, we shifted our goal to identify applications used in civil aviation that could benefit from the former. Hence, the current effort lead by governmental agencies to define and refine UTM systems was a good candidate because the different architecture modules of proposed UTM systems are still under development. By analysing the various proposed projects and architectures, we identified that for a safe and resilient implementation, they required a data layer that different stakeholders could use. This data exchange layer must offer a high degree of trust and consistency for the data so that the information can be used as a trusted oracle by the different functioning blocks. Therefore we designed, in Chapter 5, and prototyped a tracking and position reporting system that could operate within very low-level airspace, even when deployed in densely populated areas. The prototype, tested with the help of MACE, offered a distributed data store that can be deployed in edge cloudlets throughout the cities. Having many low-range antennas reduce interference, and the fast key-value replication system allows data to be available with low latency (under 500ms for full replication) in the entire coverage area. Since the system foresees an open API, it can be used by different applications. The low latency allows the system also to be used for emerging UTM services, such as Distributed Detect and Avoid Systems, that

can leverage strongly consistent data replication to provide distributed, reliable applications.

Considering the expected scale of the upcoming UTM systems and the expectation that they will be fully autonomous using machine-to-machine communication, it was natural to explore Distributed Detect And Avoid systems. In Chapter 6, we verified the possibility of leveraging edge-assisted computation offloading so that constrained UASs could be able to run complex algorithms and consequently be able to execute contingency manoeuvre in a timely manner. It was possible to evaluate the feasibility of such a system with a proof of concept. The optimisation algorithm yielded valid partition plans for a parallel task offloading with runtimes lower than 50ms. Considering the turnaround time to offload and run the task and the optimisation runtime, an aircraft would have enough time to manoeuvre without resorting to emergency stops.

7.2 Future Work

With the fluid model developed in our project, future work will allow research on self-stabilizing topology control mechanisms, optimized node positioning and replica placement schemes for distributed systems. The work can also be expanded to model different replication protocols. In the fluid model itself, there is the possibility of extending it to become sensitive to long and short lived traffic flows, to simulate message droppings after a maximum number of retries by modelling a probability function and to reproduce a gossip routing protocol. For the mobility, considering the models implemented for motion control, more control laws can be incorporated to increase robustness. Moreover, optimization techniques can be used to automatically tune the controllers' gains.

The currently implemented motion controls can already be used for the DDAA application, and the proof-of-concept can be transferred to work together with our fluid model, creating a DDAA simulator with realistic scenarios based on existing datasets. The algorithm for calculating new flight paths based on potential fields can be made parallel since it is based on the tessellation of the surroundings, which can be split into different sizes, which requires only border synchronisation.

For future work on MACE, to overcome the scalability barrier, we can leverage the built-in functions for distributed emulation available in both CORE and EMANE. Moreover, as future research directions on the position tracking data layer, we plan to research resource allocation strategies to reduce the number of replicas while keeping a highly available UTM service and full area coverage, as well as an automatic replica placement and migration algorithm that can cope with the UTM dynamicity.

List of Figures

List of Tables

Bibliography

[1] Mainak Adhikari, Satish Narayana Srirama, and Tarachand Amgoth. "Application Offloading Strategy for Hierarchical Fog Environment Through Swarm Optimization". In: *IEEE Internet of Things Journal* 7.5 (2020), pp. 4317–4328. ISSN: 23274662. DOI: 10.1109/JIOT.2019.2958400.

[2] C. Adjih et al. "FIT IoT-LAB: A large scale open experimental IoT testbed". In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, pp. 459–464. DOI: 10.1109/WF-IoT.2015.7389098.

[3] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. "Dissecting the performance of strongly-consistent replication protocols". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2019). Citation Key: Ailijiang2019, pp. 1696–1710. ISSN: 07308078. DOI: 10.1145/3299869.3319893.

[4] J. N. Al-Karaki and A. E. Kamal. "Routing techniques in wireless sensor networks: a survey". In: *IEEE Wireless Communications* 11.6 (2004), pp. 6–28. DOI: 10.1109/MWC.2004.1368893.

[5] Dario Albani et al. "Dynamic UAV Swarm Deployment for Non-Uniform Coverage". In: *Ifaamas* 9.Aamas (2018), pp. 523–531.

[6] D. Alejo et al. "Collision-free trajectory planning based on Maneuver Selection-Particle Swarm Optimization". In: *2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015* (2015), 72–81. DOI: 10.1109/ICUAS.2015.7152277.

[7] Safwan Alwan et al. "A Scalable Scheme for Joint Routing and Resource Allocation in LTE-D2D Based Offloading". In: *MSWiM 2020 - Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (2020), pp. 261–265. DOI: 10.1145/3416010.3423243.

[8] Matthew Ayamga, Selorm Akaba, and Albert Apotele Nyaaba. "Multifaceted applicability of drones: A review". In: *Technological Forecasting and Social Change* 167.February (2021). Citation Key: Ayamga2021, p. 120677. ISSN: 00401625. DOI: 10.1016/j.techfore.2021.120677.

[9] M Mahdi Azari et al. "Evolution of Non-Terrestrial Networks From 5G to 6G : A Survey". In: (), pp. 1–35. arXiv: arXiv:2107.06881v1.

[10] Joshua Baculi and Corey Ippolito. "Onboard decision-making for nominal and contingency suas flight". In: *AIAA Scitech 2019 Forum* January (2019), pp. 1–16. DOI: 10.2514/6.2019-1457.

[11] Alessandra Bagnato et al. "Designing Swarms of Cyber-Physical Systems". In: (2017), pp. 305–312. DOI: 10.1145/3075564.3077628.

[12] Alessandra Bagnato et al. "Designing Swarms of Cyber-Physical Systems: The H2020 CPSwarm Project: Invited Paper". In: *Proceedings of the Computing Frontiers Conference*. CF'17. Siena, Italy: Association for Computing Machinery, 2017, pp. 305–312. ISBN: 9781450344876. DOI: 10.1145/3075564.3077628. URL: https://doi.org/10.1145/3075564.3077628.

[13] Vaibhav Bajpai et al. "The dagstuhl beginners guide to reproducibility for experimental networking research". In: *Computer Communication Review* 49.1 (2019). arXiv: 1902.02165, pp. 24–30. ISSN: 19435819. DOI: 10.1145/3314212.3314217.

[14] *B.A.T.M.A.N.* 2022 (accessed October, 2022). URL: https://www.open-mesh.org/projects/batman-adv/wiki.

[15] Oladayo Bello and Sherali Zeadally. "Internet of underwater things communication: Architecture, technologies, research challenges and future opportunities". In: *Ad Hoc Networks* 135 (2022), p. 102933. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2022.102933. URL: https://www.sciencedirect.com/science/article/pii/S1570870522001172.

[16] Hans Van Den Berg, Michel Mandjes, and Frank Roijers. "Performance Modeling of a Bottleneck Node in an IEEE 802 . 11 Ad-Hoc Network". In: *Ad-Hoc Networks and Wireless* (2006), pp. 321–336.

[17] Juan Besada et al. "Drone flight planning for safe urban operations: UTM requirements and tools". In: *Personal and Ubiquitous Computing* (2020), pp. 924–930. ISSN: 16174917. DOI: 10.1007/s00779-019-01353-7.

[18] Matías Bonaventura and Rodrigo Castro. "Fluid-flow and packet-level models of data networks unified under a modular/hierarchical framework: Speedups and simplicity, combined". In: *Proceedings - Winter Simulation Conference*. 2019. ISBN: 9781538665725. DOI: 10.1109/WSC.2018.8632499.

[19] Roland Bouffanais. *Design and Control of Swarm Dynamics*. 2016. ISBN: 9789812877505. DOI: 10.1007/978-981-287-751-2.

[20] Gabriel Bracha and Sam Toueg. "Asynchronous Consensus and Broadcast Protocols". In: *Journal of the ACM (JACM)* 32.4 (1985). Citation Key: Bracha1985, pp. 824–840. ISSN: 1557735X. DOI: 10.1145/4221.214134.

[21] Manuele Brambilla et al. "Swarm robotics: A review from the swarm engineering perspective". In: *Swarm Intelligence* 7.1 (2013), pp. 1–41. ISSN: 19353812. DOI: 10.1007/s11721-012-0075-2.

[22] Brunobcfum. *Brunobcfum/pymace: Python mobile AH-hoc computing emulator*. URL: https://github.com/brunobcfum/pymace.

[23] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer Berlin Heidelberg, 2011. ISBN: 9783642152603. URL: https://books.google.fr/books?id=Y8lHVFeJ6EQC.

[24] Rodrigo N. Calheiros et al. “CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms”. In: *Softw. Pract. Exper.* 41.1 (Jan. 2011), pp. 23–50. ISSN: 0038-0644. DOI: 10.1002/spe.995. URL: https://doi.org/10.1002/spe.995.

[25] Baptiste Cecconi et al. “NOIRE study report: Towards a low frequency radio interferometer in space”. In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–19. DOI: 10.1109/AERO.2018.8396742.

[26] Anjan Chakrabarty and Corey Ippolito. “Autonomous flight for multi-copters flying in UTM -TCL4+ sharing common airspace”. In: *AIAA Scitech 2020 Forum* 1 PartF.January (2020). DOI: 10.2514/6.2020-0881.

[27] Anjan Chakrabarty et al. “Real-time path planning for multi-copters flying in UTM-TCL4”. In: *AIAA Scitech 2019 Forum* January (2019). DOI: 10.2514/6.2019-0958.

[28] J. Chen et al. “Assessing Distributed Consensus Performance on Mobile Cyber-Physical System Swarms”. In: *International Wireless Communications and Mobile Computing Conference (IWCMC 2023)* (2023).

[29] Jienan Chen et al. “An intelligent task offloading algorithm (iTOA) for UAV edge computing network”. In: *Digital Communications and Networks* April (2020). ISSN: 23528648. DOI: 10.1016/j.dcan.2020.04.008. URL: https://doi.org/10.1016/j.dcan.2020.04.008.

[30] Marco Cicala et al. “Scalable distributed state estimation in UTM context”. In: *Sensors (Switzerland)* 20.9 (2020), pp. 1–20. ISSN: 14248220. DOI: 10.3390/s20092682.

[31] *CORE - Common Open Research Emulator*. 2022 (accessed October, 2022). URL: https://coreemu.github.io/core/.

[32] Rodolfo W.L. Coutinho et al. “Underwater wireless sensor networks: A new challenge for topology control-based systems”. In: *ACM Computing Surveys* 51.1 (2018). Citation Key: Coutinho2018. ISSN: 15577341. DOI: 10.1145/3154834.

[33] Rodolfo W.L. Coutinho et al. “Underwater wireless sensor networks: A new challenge for topology control-based systems”. In: *ACM Computing Surveys* 51.1 (2018). ISSN: 15577341. DOI: 10.1145/3154834.

[34] Alexandra De Cecco. “Modélisation Fluide de Réseaux”. Theses. UNIVERSITE DE TOULOUSE ; UT3, June 2016. URL: https://hal.archives-ouvertes.fr/tel-01486585.

[35] E. Dekens, S. Engelen, and R. Noomen. “A satellite swarm for radio astronomy”. In: *Acta Astronautica* 102 (2014), pp. 321–331. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2013.11.033. URL: https://www.sciencedirect.com/science/article/pii/S0094576513004360.

[36] Bradley Denby and Brandon Lucia. “Orbital edge computing: Nanosatellite constellations as a new class of computer system”. In: *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS* (2020), pp. 939–954. DOI: 10.1145/3373376.3378473.

[37] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numer. Math.* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[38] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the Presence of Partial Synchrony." In: 35.2 (1984). Citation Key: Dwork1984, pp. 103–118.

[39] Hanan Elazhary. "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions". In: *Journal of Network and Computer Applications* 128.November 2018 (2019), pp. 105–140. ISSN: 10958592. DOI: 10.1016/j.jnca.2018.10.021. URL: https://doi.org/10.1016/j.jnca.2018.10.021.

[40] Mo Elsayed and Moataz Mohamed. "The impact of airspace regulations on unmanned aerial vehicles in last-mile operation". In: *Transportation Research Part D: Transport and Environment* 87.August (2020), p. 102480. ISSN: 13619209. DOI: 10.1016/j.trd.2020.102480. URL: https://doi.org/10.1016/j.trd.2020.102480.

[41] *EMANE Emulator*. 2022 (accessed October, 2022). URL: https://github.com/adjacentlink/emane/wiki.

[42] *etcd*. 2022 (accessed October, 2022). URL: https://etcd.io/.

[43] Fernanda Famá, Danilo F. S. Santos, and Angelo Perkusich. "Integrating an IoT Application Middleware with a Fog and Edge Computing Simulator". In: *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2020, pp. 1–6. DOI: 10.23919/SoftCOM50211.2020.9238245.

[44] D. G. Feitelson and D. Tsafrir. *Logs of real parallel workloads from production systems*. URL: https://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[45] Bruno Chianca Ferreira, Guillaume Dufour, and Guthemberg Silvestre. "A Lightweight Fluid Model for Mobile Ad hoc Distributed Systems". In: *2022 IEEE Symposium on Computers and Communications (ISCC)*. 2022, pp. 1–7. DOI: 10.1109/ISCC55528.2022.9912980.

[46] Bruno Chianca Ferreira, Guillaume Dufour, and Guthemberg Silvestre. "MACE: A Mobile Ad-hoc Computing Emulation Framework". In: *Proceedings - International Conference on Computer Communications and Networks, ICCCN* 2021-July (2021). ISSN: 10952055. DOI: 10.1109/ICCCN52240.2021.9522185.

[47] Bruno Chianca Ferreira, Guillaume Dufour, and Guthemberg Silvestre. "Towards a Novel UAV Position Tracking and Reporting System for Very Low Level Airspace". In: *ERTS2022*. Toulouse, France, June 2022. URL: https://hal.archives-ouvertes.fr/hal-03450168.

[48] Miroslav Fiedler. "Algebraic connectivity of graphs". eng. In: *Czechoslovak Mathematical Journal* 23.2 (1973), pp. 298–305. URL: http://eudml.org/doc/12723.

[49] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *J. ACM* 32.2 (1985), 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121.

[50] George H. Forman and John Zahorjan. "The Challenges of Mobile Computing". In: *Computer* 27.4 (1994), pp. 38–47. ISSN: 00189162. DOI: 10.1109/2.274999.

[51] F.∼Xue and P.∼R.∼Kumar. "The number of neighbors needed for connectivity of wireless networks". In: *Wireless Networks* 10.2 (2004), pp. 169–181.

[52] Holger Füßler et al. "MobiCom Poster: Location-Based Routing for Vehicular Ad-Hoc Networks". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 7.1 (Jan. 2003), pp. 47–49. ISSN: 1559-1662. DOI: 10.1145/881978.881992. URL: https://doi.org/10.1145/881978.881992.

[53] Matthias Függer, Thomas Nowak, and Kyrill Winkler. "On the radius of nonsplit graphs and information dissemination in dynamic networks". en. In: *Discrete Applied Mathematics* 282 (2020), pp. 257–264. ISSN: 0166218X. DOI: 10.1016/j.dam.2020.02.013.

[54] Xin Gao et al. "PORA: Predictive Offloading and Resource Allocation in Dynamic Fog Computing Systems". In: *IEEE Internet of Things Journal* 7.1 (2020). Citation Key: Gao2020 ISBN: 9781538680889, pp. 72–87. ISSN: 23274662. DOI: 10.1109/JIOT.2019.2945066.

[55] Paul Gaynor and Daniel Coore. "Towards distributed wilderness search using a reliable distributed storage device built from a swarm of miniature UAVs". In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014, pp. 596–601. DOI: 10.1109/ICUAS.2014.6842302.

[56] Cinara Ghedini et al. "Improving robustness in multi-robot networks". In: *IFAC-PapersOnLine* 48.19 (2015), pp. 63–68. ISSN: 24058963. DOI: 10.1016/j.ifacol.2015.12.011. URL: http://dx.doi.org/10.1016/j.ifacol.2015.12.011.

[57] Seth Gilbert and Nancy Lynch. "Perspectives on the CAP Theorem". In: *Computer* 45.2 (2012). Citation Key: Gilbert2012, pp. 30–36. ISSN: 0018-9162. DOI: 10.1109/mc.2011.389.

[58] *GNS3 Network Emulator*. 2022 (accessed October, 2022). URL: https://www.gns3.com/.

[59] C. Goerzen, Z. Kong, and B. Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance*. Vol. 57. 1–4. 2010. ISBN: 1084600993831. DOI: 10.1007/s10846-009-9383-1.

[60] Shushi Gu et al. "Grouping-based consistency protocol design for end-edge-cloud hierarchical storage system". In: *IEEE Access* 8.Cc (2020), pp. 8959–8973. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2964626.

[61] Rachid Guerraoui, Jovan Komatovic, and Dragos-Adrian Seredinschi. "Dynamic Byzantine Reliable Broadcast [Technical Report]". In: (2020). arXiv: 2001.06271. URL: http://arxiv.org/abs/2001.06271.

[62] Rachid Guerraoui et al. "Scalable Byzantine reliable broadcast". In: *Leibniz International Proceedings in Informatics, LIPIcs* 146.22 (2019), pp. 1–16. ISSN: 18688969. DOI: 10.4230/LIPIcs.DISC.2019.22.

[63] Yan Guo et al. “User allocation-aware edge cloud placement in mobile edge computing”. In: *Software: Practice and Experience* 50.5 (2020), pp. 489–502. DOI: https://doi.org/10.1002/spe.2685. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2685.

[64] Harshit Gupta et al. “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments”. In: *Software: Practice and Experience* 47.9 (2017), pp. 1275–1296. DOI: https://doi.org/10.1002/spe.2509. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2509. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2509.

[65] J. Hasenburg et al. “MockFog: Emulating Fog Computing Infrastructure in the Cloud”. In: *2019 IEEE International Conference on Fog Computing (ICFC)*. 2019, pp. 144–152. DOI: 10.1109/ICFC.2019.00026.

[66] *INET Framework Documentation - Network Emulation*. (accessed October, 2022). URL: https://inet.omnetpp.org/docs/users-guide/ch-emulation.html.

[67] Iswanto et al. “Artificial potential field algorithm implementation for quadrotor path planning”. In: *International Journal of Advanced Computer Science and Applications* 10.8 (2019), 575–585. ISSN: 21565570. DOI: 10.14569/ijacsa.2019.0100876.

[68] Sam Ade Jacobs et al. “A scalable distributed RRT for motion planning”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2013). Citation Key: Jacobs2013 ISBN: 9781467356411, 5088–5095. ISSN: 10504729. DOI: 10.1109/ICRA.2013.6631304.

[69] P. Jacquet et al. “Optimized link state routing protocol for ad hoc networks”. In: *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.* 2001, pp. 62–68. DOI: 10.1109/INMIC.2001.995315.

[70] Yang Jian and Christian Poellabauer. “SATSS : A Self-Adaptive Task Scheduling Scheme for Mobile Edge Computing”. In: *International Conference on Computer Communications and Networks*. 2021. ISBN: 978-0-7381-1330-2.

[71] Sertac Karaman et al. “Anytime motion planning using the RRT”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2011), 1478–1483. ISSN: 10504729. DOI: 10.1109/ICRA.2011.5980479.

[72] A. Kato, M. Takai, and S. Ishihara. “Design and implementation of a wireless network tap device for IEEE 802.11 wireless network emulation”. In: *2017 Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*. 2017, pp. 1–6. DOI: 10.23919/ICMU.2017.8330098.

[73] Zahid Khan, Pingzhi Fan, and Sangsha Fang. “On the connectivity of vehicular Ad Hoc Network Under Various Mobility Scenarios”. In: *IEEE Access* 5 (2017), pp. 22559–22565. ISSN: 21693536. DOI: 10.1109/ACCESS.2017.2761551.

[74] Yoohwan Kim, Ju Yeon Jo, and Sungchul Lee. “ADS-B vulnerabilities and a security solution with a timestamp”. In: *IEEE Aerospace and Electronic Systems Magazine* 32.11 (2017), pp. 52–61. ISSN: 08858985. DOI: 10.1109/MAES.2018.160234.

[75] Oleg Kolosov et al. “Benchmarking in the dark: On the absence of comprehensive edge datasets”. In: *HotEdge 2020 - 3rd USENIX Workshop on Hot Topics in Edge Computing* (2020).

[76] Parimal Kopardekar et al. “Unmanned aircraft system traffic management (UTM) concept of operations”. In: *16th AIAA Aviation Technology, Integration, and Operations Conference* (2016), pp. 1–16.

[77] Kalmanje Krishnakumar et al. “Safe autonomous flight environment (SAFE50) for the notional last "50 ft" of operation of "55 lb" class of UAS”. In: *AIAA Information Systems-AIAA Infotech at Aerospace, 2017* (2017), pp. 1–7. DOI: 10.2514/6.2017-0445.

[78] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. “Distributed computation in dynamic networks”. en. In: *Proceedings of the forty-second ACM symposium on Theory of computing*. Cambridge Massachusetts USA: ACM, 2010, pp. 513–522. ISBN: 978-1-4503-0050-6. DOI: 10.1145/1806689.1806760. URL: https://dl.acm.org/doi/10.1145/1806689.1806760.

[79] Ching-Chi L. et al. “Cost-Effective Offloading Strategies for UAV Contingency Planning in Smart Cities”. In: *SmartCityCom Workshop - International Conference on Computer Communications and Networks (ICCCN 2023)* (2023).

[80] Nader S. Labib et al. “A multilayer low-altitude airspace model for UAV traffic management”. In: *DIVANet 2019 - Proceedings of the 9th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications* (2019), pp. 57–63. DOI: 10.1145/3345838.3355998.

[81] Leslie Lamport. “Paxos Made Simple”. In: *ACM SIGACT News* (2001). ISSN: 01635700. DOI: 10.1145/568425.568433.

[82] Thomas Lemaire, Rachid Alami, and Simon Lacroix. “A distributed tasks allocation scheme in multi-UAV context”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2004.4 (2004). Citation Key: Lemaire2004, pp. 3622–3627. ISSN: 10504729. DOI: 10.1109/robot.2004.1308816.

[83] J. Li et al. “Capacity of ad hoc wireless networks”. In: *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM* (2001), pp. 61–69. DOI: 10.1145/381677.381684.

[84] Junfei Li et al. “Bio-inspired intelligence with applications to robotics: a survey”. In: *Intelligence & Robotics* (2021). arXiv: 2206.08544. DOI: 10.20517/ir.2021.08.

[85] Mo Li, Zhenjiang Li, and Athanasios V. Vasilakos. “A Survey on Topology Control in Wireless Sensor Networks: Taxonomy, Comparative Study, and Open Issues”. In: *Proceedings of the IEEE* 101.12 (2013), pp. 2538–2557. DOI: 10.1109/JPROC.2013.2257631.

[86] Hao Liang and Weihua Zhuang. “Stochastic Modeling and Optimization in a Microgrid: A Survey”. In: *ENERGIES* 7.4 (2014), pp. 2027–2050. DOI: 10.3390/en7042027.

[87] Zhao Liang and Liang Qilian. "Hop-Distance Estimation in Wireless Sensor Networks with Applications to Resources Allocation". In: *EURASIP Journal on Wireless Communications and Networking* 2007 (May 2007). DOI: 10.1155/2007/84256.

[88] Chin E. Lin. "An ADS-B like communication for UTM". In: *Integrated Communications, Navigation and Surveillance Conference, ICNS* 2019-April (2019), pp. 1–12. ISSN: 21554951. DOI: 10.1109/ICNSURV.2019.8735199.

[89] Ying Hong Lin, Chin E. Lin, and Hsu Chan Chen. "ADS-B Like UTM surveillance using APRS infrastructure". In: *Aerospace* 7.7 (2020), pp. 1–14. ISSN: 22264310. DOI: 10.3390/AEROSPACE7070100.

[90] John D. C. Little and Stephen C. Graves. "Little's Law". In: *Building Intuition: Insights From Basic Operations Management Models and Principles*. Ed. by Dilip Chhajed and Timothy J. Lowe. Boston, MA: Springer US, 2008, pp. 81–100. ISBN: 978-0-387-73699-0. DOI: 10.1007/978-0-387-73699-0_5. URL: https://doi.org/10.1007/978-0-387-73699-0_5.

[91] Jianhui Liu and Qi Zhang. "Offloading Schemes in Mobile Edge Computing for Ultra-Reliable Low Latency Communications". In: *IEEE Access* 6 (2018), pp. 12825–12837. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2800032.

[92] Yang Liu et al. "Review on cyber-physical systems". In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (2017), pp. 27–40. ISSN: 23299274. DOI: 10.1109/JAS.2017.7510349.

[93] Emmanuel Lochin, Tanguy Pérennou, and Laurent Dairaine. "When should I use network emulation?" en. In: *annals of telecommunications - annales des télécommunications* 67.5-6 (2012), pp. 247–255. ISSN: 0003-4347, 1958-9395. DOI: 10.1007/s12243-011-0268-5.

[94] Jonas Lundberg, Karljohan Lundin Palmerius, and Billy Josefsson. "Urban Air Traffic Management (UTM) Implementation in cities - Sampled side-effects". In: *AIAA/IEEE Digital Avionics Systems Conference - Proceedings* 2018-Septe.September (2018). ISSN: 21557209. DOI: 10.1109/DASC.2018.8569869.

[95] Matin MacKtoobian, Denis Gillet, and Jean Paul Kneib. "Astrobotics: Swarm Robotics for Astrophysical Studies". In: *IEEE Robotics and Automation Magazine* 28.3 (2021), pp. 92–101. ISSN: 1558223X. DOI: 10.1109/MRA.2020.3044911.

[96] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. "Fog Computing: A taxonomy, survey and future directions". In: *Internet of Things* 0.9789811058608 (2018). arXiv: 1611.05539 Citation Key: Mahmud2018 ISBN: 9789811058615, pp. 103–130. ISSN: 21991081. DOI: 10.1007/978-981-10-5861-5_5.

[97] Emerson A Marconato et al. "AVENS - A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System". In: (2017), pp. 6275–6284.

[98] A. Markus and A. Kertesz. "A survey and taxonomy of simulation environments modelling fog computing". In: *Simulation Modelling Practice and Theory* 101 (2020). cited By 1. DOI: 10.1016/j.simpat.2019.102042.

[99] M. A. Marsan et al. "Using partial differential equations to model TCP mice and elephants in large IP networks". In: *IEEE INFOCOM 2004*. Vol. 4. 2004, 2821–2832 vol.4. DOI: 10.1109/INFCOM.2004.1354699.

[100] M. Ajmone Marsan et al. "Using partial differential equations to model TCP mice and elephants in large IP networks". In: *Proceedings - IEEE INFOCOM*. 2004. ISBN: 0780383559. DOI: 10.1109/INFCOM.2004.1354699.

[101] R. Mayer et al. "EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures". In: *2017 IEEE Fog World Congress (FWC)*. 2017, pp. 1–6. DOI: 10.1109/FWC.2017.8368525.

[102] Tim McCarthy, Lars Pforte, and Rebekah Burke. "Fundamental Elements of an Urban UTM". In: *Aerospace* 7.7 (2020). ISSN: 2226-4310. DOI: 10.3390/aerospace7070085. URL: https://www.mdpi.com/2226-4310/7/7/85.

[103] Marco Minelli et al. "Self-optimization of resilient topologies for fallible multi-robots". In: *Robotics and Autonomous Systems* 124 (2020), p. 103384. ISSN: 09218890. DOI: 10.1016/j.robot.2019.103384. URL: https://doi.org/10.1016/j.robot.2019.103384.

[104] Franco Minucci, Evgenii Vinogradov, and Sofie Pollin. "Avoiding Collisions at Any (Low) Cost: ADS-B like Position Broadcast for UAVs". In: *IEEE Access* 8 (2020), pp. 121843–121857. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3007315. arXiv: 2003.13499.

[105] J. Miranda et al. "Path loss exponent analysis in Wireless Sensor Networks: Experimental evaluation". In: *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. 2013, pp. 54–58. DOI: 10.1109/INDIN.2013.6622857.

[106] Seongin Na et al. "Bio-inspired artificial pheromone system for swarm robotics applications". In: *Adaptive Behavior* 29.4 (2021), pp. 395–415. ISSN: 17412633. DOI: 10.1177/1059712320918936.

[107] M. I. Naas et al. "An Extension to iFogSim to Enable the Design of Data Placement Strategies". In: *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. 2018, pp. 1–8. DOI: 10.1109/CFEC.2018.8358724.

[108] A. Narayanasamy, Y. A. Ahmad, and M. Othman. "Nanosatellites constellation as an IoT communication platform for near equatorial countries". In: *IOP Conference Series: Materials Science and Engineering* 260.1 (2017). ISSN: 1757899X. DOI: 10.1088/1757-899X/260/1/012028.

[109] Jer Shyuan Ng et al. "Joint Auction-Coalition Formation Framework for Communication-Efficient Federated Learning in UAV-Enabled Internet of Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 22.4 (2021), pp. 2326–2344. DOI: 10.1109/TITS.2020.3041345.

[110] *NS3 - Emulation Overview*. 2022 (accessed October, 2022). URL: https://www.nsnam.org/docs/release/3.31/models/html/emulation-overview.html.

[111] R. S. Oliver and G. Fohler. "Probabilistic estimation of end-to-end path latency in Wireless Sensor Networks". In: *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. 2009, pp. 423–431. DOI: 10.1109/MOBHOC.2009.5336970.

[112] Gregory L. Orrell, Angela Chen, and Christopher J. Reynolds. "Small unmanned aircraft system (SUAS) automatic dependent surveillance-broadcast (ADS-B) like surveillance concept of operations: A path forward for small UAS surveillance". In: *AIAA/IEEE Digital Avionics Systems Conference - Proceedings* 2017-Septe (2017). ISSN: 21557209. DOI: 10.1109/DASC.2017.8102026.

[113] *OSMNX 1.3.0¶*. URL: https://osmnx.readthedocs.io/en/stable/.

[114] Omar Sami Oubbati et al. "Routing in flying Ad Hoc networks: Survey, constraints, and future challenge perspectives". In: *IEEE Access* 7 (2019), pp. 81057–81105. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2923840.

[115] Brian Paden et al. "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016). arXiv: 1604.07446, 33–55. ISSN: 23798858. DOI: 10.1109/TIV.2016.2578706.

[116] Jong Hong Park et al. "Unmanned Aerial System Traffic Management with WAVE Protocol for Collision Avoidance". In: *International Conference on Ubiquitous and Future Networks, ICUFN* 2018-July (2018), pp. 8–10. ISSN: 21658536. DOI: 10.1109/ICUFN.2018.8436836.

[117] Marcelo Pereyra et al. "A Survey of Stochastic Simulation and Optimization Methods in Signal Processing". In: *IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING* 10.2 (2016), pp. 224–241. ISSN: 1932-4553. DOI: 10.1109/JSTSP.2015.2496908.

[118] C. E. Perkins and E. M. Royer. "Ad-hoc on-demand distance vector routing". In: *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*. 1999, pp. 90–100. DOI: 10.1109/MCSA.1999.749281.

[119] Radan Pesic. *New edition of the U-space ConOps published by the CORUS-XUAM project team*. accessed Mars, 2023. URL: https://corus-xuam.eu/new-u-space-conops/.

[120] André Platzer. *Logical foundations of cyber-physical systems*. 2018. ISBN: 978-3-319-63588-0. DOI: 10.1007/978-3-319-63588-0.

[121] Pasu Poonpakdee and Giuseppe Di Fatta. "Robust and efficient membership management in large-scale dynamic networks". en. In: *Future Generation Computer Systems* 75 (2017), pp. 85–93. ISSN: 0167739X. DOI: 10.1016/j.future.2017.02.033.

[122] Lingjun Pu et al. "D2D Fogging: An Energy-Efficient and Incentive-Aware Task Offloading Framework via Network-Assisted D2D Collaboration". In: *IEEE Journal on Selected Areas in Communications* 34.12 (2016). Citation Key: Pu2016, pp. 3887–39014. ISSN: 07338716. DOI: 10.1109/JSAC.2016.2624118.

[123] Alexander Puzicha and Peter Buchholz. "Real-Time Simulation of Robot Swarms with Restricted Communication Skills". In: *Proceedings of the 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2020* (2020). DOI: 10.1109/DS-RT50469.2020.9213618.

[124] T. Qayyum et al. "FogNetSim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment". In: *IEEE Access* 6 (2018), pp. 63570–63583. DOI: 10.1109/ACCESS.2018.2877696.

[125] Han Qi and Abdullah Gani. "Research on mobile cloud computing: Review, trend and perspectives". In: *2012 2nd International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2012* (2012), pp. 195–202. DOI: 10.1109/DICTAP.2012.6215350.

[126] Mohammadreza Radmanesh et al. "Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study". In: *Unmanned Systems* 6.2 (2018). Citation Key: Radmanesh2018, 95–118. ISSN: 23013869. DOI: 10.1142/S2301385018400022.

[127] Mohammadreza Radmanesh et al. "Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study". In: *Unmanned Systems* 6.2 (2018), pp. 95–118. ISSN: 23013869. DOI: 10.1142/S2301385018400022.

[128] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. en. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-94140-0. DOI: 10.1007/978-3-319-94141-7. URL: http://link.springer.com/10.1007/978-3-319-94141-7.

[129] Salman Raza et al. "A vehicle to vehicle relay-based task offloading scheme in Vehicular Communication Networks". In: *PeerJ Computer Science* 7 (2021). Citation Key: Raza2021, pp. 1–20. ISSN: 23765992. DOI: 10.7717/peerj-cs.486.

[130] Wei Ren and Randal W Beard. *Distributed Consensus in Multi-vehicle Cooperative Control*. 2008. ISBN: 978-1-84800-014-8.

[131] Joseph Rios et al. "Flight Demonstration of Unmanned Aircraft System (Uas) Traffic Management (utm) at Technical Capability Level 3". In: *Aiaa Aviation 2020 Forum* 1 PartF (2020). DOI: 10.2514/6.2020-2851.

[132] Joseph Rios et al. "NASA UAS traffic management national campaign: Operations across Six UAS Test Sites". In: *AIAA/IEEE Digital Avionics Systems Conference - Proceedings* 2016-December (2016), pp. 1–6. ISSN: 21557209. DOI: 10.1109/DASC.2016.7778080.

[133] Frank Roijers, Hans Van Den Berg, and Michel Mandjes. "Fluid-flow modeling of a relay node in an IEEE 802.11 wireless ad-hoc network". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4516 LNCS (2007), pp. 321–324. ISSN: 16113349. DOI: 10.1007/978-3-540-72990-7_31.

[134] Frank Roijers, Hans Van Den Berg, and Michel Mandjes. "Performance analysis of differentiated resource-sharing in a wireless ad-hoc network". In: *Performance Evaluation* 67.7 (2010), pp. 528–547. ISSN: 01665316. DOI: 10.1016/j.peva.2010.01.005. URL: http://dx.doi.org/10.1016/j.peva.2010.01.005.

[135] Rudolfs Rumba and Agris Nikitenko. "The wild west of drones: A review on autonomous-UAV traffic-management". In: *2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020* (2020), pp. 1317–1322. DOI: 10.1109/ICUAS48674.2020.9214031.

[136] Lorenzo Sabattini, Nikhil Chopra, and Cristian Secchi. "Decentralized connectivity maintenance for cooperative control of mobile robotic systems". In: *International Journal of Robotics Research* 32.12 (2013), pp. 1411–1423. ISSN: 02783649. DOI: 10.1177/0278364913499085.

[137] David Sacharny and Thomas C Henderson. "A Lane-Based Approach for Large-Scale Strategic Conflict Management for UAS Service Suppliers". In: (2019), pp. 937–945.

[138] David Sacharny, Thomas C. Henderson, and Ejay Guo. "A DDDAS Protocol for Real-Time Large-Scale UAS Flight Coordination". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12312 LNCS (2020), pp. 49–56. ISSN: 16113349. DOI: 10.1007/978-3-030-61725-7_8.

[139] Indranil Sarkar et al. "A Collaborative Computational Offloading Strategy for Latency-Sensitive Applications in Fog Networks". In: *IEEE Internet of Things Journal* 9.6 (2022), pp. 4565–4572. ISSN: 23274662. DOI: 10.1109/JIOT.2021.3104324.

[140] Indranil Sarkar et al. "Dynamic Task Placement for Deadline-Aware IoT Applications in Federated Fog Networks". In: *IEEE Internet of Things Journal* 9.2 (2022), pp. 1469–1478. ISSN: 23274662. DOI: 10.1109/JIOT.2021.3088227.

[141] Martin Saska et al. "Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles". In: *Journal of Intelligent & Robotic Systems* 84.1 (2016), pp. 469–492.

[142] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. "The computing landscape of the 21st century". In: *HotMobile 2019 - Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications* (2019), pp. 45–50. DOI: 10.1145/3301293.3302357.

[143] Mahadev Satyanarayanan et al. "The Role of Edge Offload for Hardware-Accelerated Mobile Devices". In: *HotMobile 2021 - Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (2021), pp. 22–29. DOI: 10.1145/3446382.3448360.

[144] N. Schmidt et al. "MiniWorld: Resource-aware distributed network emulation via full virtualization". In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. July 2017, pp. 818–825. DOI: 10.1109/ISCC.2017.8024628.

[145] Fred B. Schneider. "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial". In: *ACM Computing Surveys (CSUR)* 22.4 (1990), pp. 299–319. ISSN: 15577341. DOI: 10.1145/98163.98167.

[146] Melanie Schranz et al. "Modelling a CPS swarm system: A simple case study". In: *MODELSWARD 2018 - Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development* 2018-Janua.Modelsward (2018), pp. 615–624. DOI: 10.5220/0006731106150624.

[147] Drew Scott et al. “Distributed bidding-based detect-and-avoid for multiple unmanned aerial vehicles in national airspace”. In: *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019*. 2019, pp. 930–936. ISBN: 9781728103327. DOI: 10.1109/ICUAS.2019.8798215.

[148] Hazim Shakhatreh et al. “Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges”. In: *Ieee Access* 7 (2019), pp. 48572–48634.

[149] Ting Shi et al. “An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds”. In: *Pervasive and Mobile Computing* 27 (2016). Citation Key: Shi2016, pp. 90–105. ISSN: 15741192. DOI: 10.1016/j.pmcj.2015.07.005.

[150] Mauricio J. Silva. et al. “Temporal Evolution of Vehicular Network Simulators: Challenges and Perspectives”. In: *Proceedings of the 20th International Conference on Enterprise Information Systems - Volume 2: ICEIS,* INSTICC. SciTePress, 2018, pp. 51–60. ISBN: 978-989-758-298-1. DOI: 10.5220/0006696900510060.

[151] and Single European Sky ATM Research 3 Joint Undertaking. *European drones outlook study : unlocking the value for Europe*. Publications Office, 2017. DOI: doi/10.2829/085259.

[152] Michal Skowron et al. “Sense and avoid for small unmanned aircraft systems: Research on methods and best practices”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 233 (16 2019), pp. 6044–6062. ISSN: 20413025. DOI: 10.1177/0954410019867802.

[153] B. Sliwa, S. Falten, and C. Wietfeld. “Performance Evaluation and Optimization of B.A.T.M.A.N. V Routing for Aerial and Ground-Based Mobile Ad-Hoc Networks”. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. 2019, pp. 1–7. DOI: 10.1109/VTCSpring.2019.8746361.

[154] Nikki Sonenberg and Peter G. Taylor. “Networks of interacting stochastic fluid models with infinite and finite buffers”. In: *Queueing Systems* 92.3-4 (2019), pp. 293–322. ISSN: 15729443. DOI: 10.1007/s11134-019-09619-w. URL: https://doi.org/10.1007/s11134-019-09619-w.

[155] David St-Onge et al. “Planetary exploration with robot teams: Implementing higher autonomy with swarm intelligence”. In: *IEEE Robotics and Automation Magazine* 27.2 (2020), pp. 159–168. ISSN: 1558223X. DOI: 10.1109/MRA.2019.2940413.

[156] Nathan Stacey and Simone D’Amico. “Autonomous swarming for simultaneous navigation and asteroid characterization”. In: *AAS/AIAA Astrodynamics Specialist Conference*. Vol. 1. 2018.

[157] Thomas Staub, Reto Gantenbein, and Torsten Braun. “VirtualMesh: An Emulation Framework for Wireless Mesh and Ad Hoc Networks in OMNeT++”. In: *Simulation* 87.1-2 (2011), pp. 66–81. ISSN: 0037-5497. DOI: 10.1177/0037549710373909. URL: https://doi.org/10.1177/0037549710373909.

[158] Thomas Staub, Reto Gantenbein, and Torsten Braun. “VirtualMesh: An emulation framework for wireless mesh and ad hoc networks in OMNeT++”. In: *Simulation* 87.1-2 (2011), pp. 66–81. ISSN: 00375497. DOI: 10.1177/0037549710373909.

[159] Maarten van Steen and Andrew S. Tanenbaum. *Distributed systems*. en. Third edition, Version 3.01. Erscheinungsort nicht ermittelbar: Maarten van Steen, 2017. ISBN: 978-1-5430-5738-6.

[160] Daniel H. Stolfi et al. "A competitive Predator-Prey approach to enhance surveillance by UAV swarms". In: *Applied Soft Computing* 111 (2021), p. 107701. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2021.107701. URL: https://www.sciencedirect.com/science/article/pii/S1568494621006220.

[161] P. B. Sujit and R. Beard. "Multiple UAV path planning using anytime algorithms". In: *2009 American Control Conference*. 2009, pp. 2978–2983. DOI: 10.1109/ACC.2009.5160222.

[162] Qingyu Tan et al. "Evolutionary optimization-based mission planning for UAS traffic management (UTM)". In: *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019* (2019), pp. 952–958. DOI: 10.1109/ICUAS.2019.8798078.

[163] *The Grid Workloads Archive*. URL: http://gwa.ewi.tudelft.nl/datasets.

[164] Zhao Tong et al. "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment". In: *Information Sciences* 537 (2020), pp. 116–131. ISSN: 00200255. DOI: 10.1016/j.ins.2020.05.057. URL: https://doi.org/10.1016/j.ins.2020.05.057.

[165] Animesh Trivedi et al. "Sharing and Caring of Data at the Edge". In: *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, June 2020. URL: https://www.usenix.org/conference/hotedge20/presentation/trivedi.

[166] Sahil Vashisht, Sushma Jain, and Gagangeet Singh Aujla. "MAC protocols for unmanned aerial vehicle ecosystems: Review and challenges". In: *Computer Communications* 160.April (2020), pp. 443–463. ISSN: 1873703X. DOI: 10.1016/j.comcom.2020.06.011. URL: https://doi.org/10.1016/j.comcom.2020.06.011.

[167] Deepak Vasisht et al. "FarmBeats: An IoT Platform for Data-Driven Agriculture FarmBeats: An IoT Platform for Data-Driven Agriculture". In: (2017). URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vasisht.

[168] *Veins The open source vehicular network simulation framework.* (accessed October, 2022). URL: https://veins.car2x.org/.

[169] Titouan Verdu, Gautier Hattenberger, and Simon Lacroix. "Flight patterns for clouds exploration with a fleet of UAVs". In: *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019* (2019). Citation Key: Verdu2019, pp. 231–237. DOI: 10.1109/ICUAS.2019.8797953.

[170] Shangguang Wang et al. "Edge server placement in mobile edge computing". In: *Journal of Parallel and Distributed Computing* 127 (2019), pp. 160–168. ISSN: 07437315. DOI: 10.1016/j.jpdc.2018.06.008. URL: https://doi.org/10.1016/j.jpdc.2018.06.008.

[171] Xiaohui Wang et al. “Design of agile satellite constellation based on hybrid-resampling particle swarm optimization method”. In: *Acta Astronautica* 178 (2021), pp. 595–605. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2020.09.040. URL: https://www.sciencedirect.com/science/article/pii/S0094576520305786.

[172] Lei Yang et al. “Multi-UAV-Enabled Load-Balance Mobile-Edge Computing for IoT Networks”. In: *IEEE Internet of Things Journal* 7.8 (2020), pp. 6898–6908. ISSN: 23274662. DOI: 10.1109/JIOT.2020.2971645.

[173] Peng Yang et al. “Decentralized estimation and control of graph connectivity in mobile sensor networks”. In: *2008 American Control Conference*. 2008, pp. 2678–2683. DOI: 10.1109/ACC.2008.4586897.

[174] Huang Yao, Rongjun Qin, and Xiaoyu Chen. “Unmanned Aerial Vehicle for Remote Sensing Applications—A Review”. en. In: *Remote Sensing* 11.12 (2019), p. 1443. ISSN: 2072-4292. DOI: 10.3390/rs11121443.

[175] Ibrar Yaqoob et al. “Mobile ad hoc cloud: A survey”. In: *Wireless Communications and Mobile Computing* (2016). ISSN: 15308677. DOI: 10.1002/wcm.2709.

[176] Ashkan Yousefpour et al. “All one needs to know about fog computing and related edge computing paradigms: A complete survey”. In: *Journal of Systems Architecture* 98.February (2019), pp. 289–330. ISSN: 13837621. DOI: 10.1016/j.sysarc.2019.02.009. arXiv: 1808.05283. URL: https://doi.org/10.1016/j.sysarc.2019.02.009.

[177] Bo Yu, Cheng Zhong Xu, and Bin Xiao. “Detecting Sybil attacks in VANETs”. In: *Journal of Parallel and Distributed Computing* 73.6 (2013), pp. 746–756. ISSN: 07437315. DOI: 10.1016/j.jpdc.2013.02.001.

[178] Yucong Lin and S. Saripalli. “Sense and avoid for Unmanned Aerial Vehicles using ADS-B”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 6402–6407. DOI: 10.1109/ICRA.2015.7140098.

[179] Y. Zeng, M. Chao, and R. Stoleru. “EmuEdge: A Hybrid Emulator for Reproducible and Realistic Edge Computing Experiments”. In: *2019 IEEE International Conference on Fog Computing (ICFC)*. 2019, pp. 153–164. DOI: 10.1109/ICFC.2019.00027.

[180] Huazi Zhang et al. “Mobile Conductance in Sparse Networks and Mobility-Connectivity Tradeoff”. In: *IEEE Transactions on Wireless Communications* 15.4 (2016), pp. 2954–2965. ISSN: 15361276. DOI: 10.1109/TWC.2015.2513776.

[181] Qiuling Zou, Jingzhou (James) Yang, and Daan Liang. “Stochastic Optimization Applications for Robotics and Human Modeling - A Literature Survey”. In: *ADVANCES IN APPLIED HUMAN MODELING AND SIMULATION*. Ed. by VG Duffy. Advances in Human Factors and Ergonomics Series. 2012, pp. 43–54. ISBN: 978-1-4398-7032-7; 978-1-4398-7031-0.