UNIVERSIDADE D
COIMBRA

Pedro Alexandre Gonçalves Ribeiro

BUBBLES COMMUNICATION PLATFORM
PROOF OF CONCEPT
FOR SUPPORT OF U-SPACE SEPARATION MANAGEMENT SERVICE

VOLUME 1

Dissertação no âmbito do Mestrado em Engenharia Informática/Engenharia de Software orientada pelo Professor Doutor Tiago Cruz e Professor Doutor Paulo Simões e apresentada à Faculdade de Ciências e Tecnologia/Departamento de Engenharia Informática.

Junho de 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

BUBBLES communication platform proof of concept

for support of U-Space Separation Management Service

Pedro Alexandre Gonçalves Ribeiro

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Tiago Cruz and Prof. Paulo Simões and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

June 2021

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

A drone, also referred to as an Unmanned Aerial Vehicle (UAV) or Unmanned Aircraft System (UAS), constitutes an aircraft that can either fly autonomously or be remotely controlled by human input (as it is often the case for Visual Line-of-Sight control).

These systems have been growing in popularity over the past years, with many considering that we are near the inflection point that will unleash the explosion of commodity UAV-supported services. For this to be accomplished, there is a need for a new set of services and infrastructure capable of deploying multiple drones and maintaining control over what they are doing. As such, there is the need to develop a supporting infrastructure capable of making sure that they operate efficiently and safely - that is the objective of BUBBLES.

BUBBLES aims to provide invisible bubbles for drones to fly safely and efficiently, avoiding dangerous collision incidents. To ensure proper separation between UAVs and also other kinds of traffic, BUBBLES will need to get drone telemetry information during its mission continuously. For this to happen, there needs to be a communication platform connecting the drones and services offered by BUBBLES.

This thesis describes the work done into building the communication platform and the work into a test environment to help validate BUBBLE's vision. The development goals aim to propose and develop a proof of concept capable of handling all the traffic even in the most challenging situations.

The validation goals provide the environment to test the solution in different scenarios with different levels of risk to verify that the BUBBLES vision can be achieved.

Keywords

Drones Communications; AMQP; MQTT; Kafka; DDS; MAVLink; Message broker architecture; Drones Network; IoT Communications; IoT Brokers; Drone Communication System; Drone Simulation; PX4;

This page is intentionally left blank.

Resumo

Um drone, também referido como um Veículo Aéreo Não Tripulado ou Sistema de Aviação Não Tripulada, constitui uma aeronave que pode voar autonomamente ou ser controlada remotamente por um humano (como é frequentemente no controlo em casos em que o drone está na linha de visão).

Estes sistemas têm vindo a crescer em popularidade nos últimos anos, sendo que muitos consideram que estamos perto do ponto de viragem que provocará a explosão dos serviços apoiados por drones. Para que isto seja conseguido, é necessário um novo conjunto de serviços e infra-estruturas capazes de suportar vários drones e manter o controlo sobre o que eles estão a fazer. Como tal, existe a necessidade de desenvolver uma infra-estrutura de apoio capaz de garantir o seu funcionamento eficiente e seguro - esse é o objectivo do BUBBLES.

BUBBLES visa criar bolhas invisíveis para que os drones voem com segurança e eficiência, evitando incidentes perigosos de colisão. Para assegurar uma separação adequada entre UAVs e também outros tipos de tráfego, BUBBLES precisará de obter informações de telemetria das aeronaves, durante as suas missões, de forma contínua. Para que isto aconteça, é necessário que exista uma plataforma de comunicação que ligue os drones e os serviços oferecidos pelo BUBBLES.

Esta tese descreve o trabalho feito na construção da plataforma de comunicação e o trabalho feito no ambiente de teste para ajudar a validar a visão do BUBBLES. Os objectivos de desenvolvimento visam propor e desenvolver uma prova de conceito capaz de lidar com todo o tráfego, mesmo nas situações mais difíceis.

Os objectivos de validação fornecem o ambiente para testar a solução em diferentes cenários com diferentes níveis de risco para verificar que a visão BUBBLES pode ser alcançada.

Palavras-Chave

Comunicação com Drones; AMQP; MQTT; Kafka; DDS; MAVLink; Arquitetura do Mediador de Mensagens; Redes de Drones; Comunicações em IoT; Medidores de Mensagens em IoT; Sistemas de comunicação de drones; Simulação de drones; PX4;

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

ACL Access Control List.

AMQP Advanced Message Queuing Protocol.

ATC Air Traffic Control.

ATM Air Traffic Management.

BER Bit Error Rate.

BUBBLES Building Basic Blocks for a U-Space Separation Management Service.

ConOps Concept of Operations.

CORUS Concept of Operation for European UTM Sytems.

DDS Data Distribution Service.

EASA European Union Aviation Safety Agency.

EUROCAE European Organisation for Civil Aviation Equipment.

H2020 Horizon 2020.

HITL Hardware in the Loop.

ICAO Civil Aviation Organization.

IoT Internet of Things.

JARUS Joint Authorities for Rulemaking on Unmanned Systems.

LDACS L-band Digital Aeronautical Communications System.

M2M Machine to Machine.

MEDUSA Methodology for the U-Space Safety Assessment.

MQTT Message Queuing Telemetry Transport.

OMG Object Management Group.

QoS Quality of Service.

RPAS Remotely Piloted Aircraft Systems.

RPC Remote Procedure Call.

RTPS Real-Time Publish-Subscribe.

SASL Simple Authentication and Security Layer.

SES Single European Sky.

SESAR Single European Sky ATM Research.

SESAR JU Single European Sky ATM Research Joint Undertaking.

SITL Software in the Loop.

SORA Specific Operation Risk Assessment.

SPI Service Plugin Interface.

TLS Transport Layer Security.

TRL Technology Readiness Level.

UAS Unmanned Aerial Systems.

VLL Very low level airspace.

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

In 2014 the European Commission stated that the "use of remotely piloted aircraft systems" was the "new era of aviation" [24]. Remotely Piloted Aircraft Systems (RPAS) are part of a broader category of vehicles called Unmanned Aerial Systems (UAS).

More commonly called a drone, a UAS is just an aircraft without a human pilot aboard. UASs can operate with various degrees of autonomy: either under remote control by a human operator or autonomously (semi or fully) with the help of an onboard computer. These drones are the next step in aerial transportation, but their full potential will not be used without the necessary rules, guidelines, and precautions.

Air vehicles remotely controlled or autonomously flying introduce many risks. These risks involve incidents/accidents between UAS and other UAS, people, or buildings. Another concern is the use of UAS by bad-intended people to attack or provoke damage. When talking about drones, the risk involved with using them must be taken into consideration. Drones and services that provide functionalities should all be protected, especially in how the entities communicate. Transportation of information must guarantee a certain degree of security and performance to deal with human and non-human threats.

UAS is not something new, specially RPAS, but there was no environment where people could used drones safely and efficiently at the time of their firsts appearances.

They are coming to the spotlight due to three essential revolutions; the first is technological. As time went by, the search for this kind of technology kept growing and growing. What started mainly as a military-focused tool slowly grew to a myriad of applications, from agriculture to entertainment. Today we can classify the use of drones into four different groups [25]: military, enterprise, consumer, and leisure. The number of different uses for drones was achievable to the constantly evolving technology and led to this vast market for commercial drones like agriculture, imaging, delivery, racing, and transport.

The second one is social. As we reached the 21st century, the public attention to drones has snowballed, and it is expected that it will keep on growing. A 2016 European Drone Study [26] stated that leisure drones have gotten a strong growth, with "over 100% per annum in the past few years".

The third one is at a regulatory level. Due to the increased demand, we reach a time where we need to regulate the use of these drones to guarantee the safety of everyone evolved. The UE approved a new regulation on UAS operations in 2019 [27].

UASs are not far from being incorporated into our everyday lives, and the need to create a safe environment for these machines is imminent. Right now, we are in a situation where

we do not have the necessary tools, services, and related infrastructure that could support a significant number of drones in flying safely in the European sky.

To allow UASs to fill the sky with many possible objectives in mind is a difficult task. As of today, multiple organizations are making an effort to achieve this [28]. Organizations include non-extensively, Single European Sky ATM Research (SESAR), Air Traffic Control (ATC), European Organisation for Civil Aviation Equipment (EUROCAE), Civil Aviation Organization (ICAO) and European Union Aviation Safety Agency (EASA). Their objective is to harness the technology's full potential by creating and transforming the current manned aviation infrastructure to support drones and their operations.

1.1 Context

SESAR was created in 2004. Its objective was to fullfil the role as the technological pillar of Single European Sky (SES). SES is an initiative by the European Commission to reform the European Air Traffic Management (ATM) architecture.

SESAR's purpose is defining, developing, and deploying what is necessary to increase ATM performance and build Europe's intelligent air transport system through innovative technological and operational ATM solutions. Single European Sky ATM Research Joint Undertaking (SESAR JU) was set in 2007 to manage the SESAR public-private partnership. Today SESAR JU includes 19 members, representing over 100 organizations and uniting around 3000 experts in Europe and beyond.

In 2015 the European Commission appointed SESAR JU to create a blueprint on how to use UAS in a safe, secure and efficient way. The work of SESAR JU led to the concept of U-Space.

1.1.1 U-Space

U-Space [29] is an "ecosystem" designed to enable drone operations and to facilitate any routine mission in all classes of airspace and all types of environments (even the most congested). All this while addressing an appropriate interface with manned aviation and air traffic control. For this to happen, it will offer different services, dependencies, data repositories, and data flows, as well as providing a set of rules and guidelines for UAS to follow.

U-space services are divided into four sets of services [29]:

- U1 services are the U-space foundation services. They concentrate on the most basic functionalities that are e-registration, e-identification, and geofencing. These are already available.
- U2 services are the U-space initial services for drone operations management. They include flight planning, flight approval, tracking, and connecting with conventional air traffic control. With the initially defined services, these are the main block of services. Although not currently available, they are considered to be technically possible today.
- U3 services belong to the U-space advanced services. These services will be supporting more complex operations like flights in more dense areas by assisting in conflict detection and automated detection and avoidance functionalities.

- U4 services are the U-space full services. They will offer very high levels of automation, connectivity, and digitalization for both the drone and the U-Space System.

These services will provide support for UAS missions on the European sky. Since UAS failure results in damage to infrastructures or, worse, human lives, it is considered a critical system. As with any project involving critical systems, U-Space services have many associated risks. For example, risk of accidents involving drones, people, other vehicles, or infrastructures or risk of security breaches, with attackers using vulnerabilities to perform cyberattacks, jamming, spoofing, terrorism, or carrying weapons within the UAS.

In addition to the the many difficulties associated with drones crowding our skies, U-Space also considers social acceptance and focuses on increasing it.

U-Space aims to delivers all this with the level of confidence equal to what the ATC is to manned air vehicles [27].

In 2017, the SESAR JU launched a set of Horizon 2020 (H2020) exploratory research projects in the context of U-space [28]. These exploratory research projects addressed everything needed to deploy the U-Space services. H2020 projects include, non-extensively, the concept of operations for drone missions, command and control communications, surveillance, tracking, information management, cyber-resilience, geofencing, and so forth.

Since then the projects have been finished, and in 2019 more research projects were launched. One of the winning proposal within the most recently opened calls, was the Building Basic Blocks for a U-Space Separation Management Service (BUBBLES) project.

1.1.2 BUBBLES

BUBBLES [30] is an answer to an H2020 SESAR JU U-Space program. It was proposed in the scope of the SESAR 2020 exploratory research 4 (ER4) U-Space (SESAR-ER4-31-2019 - U-space).

This project only focuses on one specific U-Space service within the third set of services, the U-Space advanced (U3) Separation Management service. Their goal is to formulate and validate its concept, defining the fundamental blocks from which this service will be built and describing how they must be assembled and operated [27].

BUBBLES's concept of the Separation Management service is, in broad terms, to create bubbles around drones to determine the minimum distance needed for efficient use of the Very low level airspace (VLL) (below 150m). To increase the safe use of the VLL airspace, the more distance between the drones, the better, and to expand the efficiency of said airspace, more drones need to be launched. These two contradict themselves, and so there needs to be a compromise between the two. BUBBLES' objective is to create those bubbles to achieve separation with reasonable levels of safety and efficiency.

In addition to drones, specific areas coincide with manned air vehicles; therefore, BUBBLES will have to consider that when developing the service. It will coordinate unmanned and manned systems in designated volumes of space attributed to U-Space, and it will coordinate the UAS in spaces attributed to the ATM.

Blocks defined by BUBBLES will need to communicate with each other and consume UAS' flight information, which is needed to create the bubbles and prevent them from touching each other; in case they touch, assist in resolving the issue. All this without disturbing manned aviation.

1.2 Motivation

UASs are still maturing as a technology, and the right "ecosystem" can accelerate this. U-Space offers what is needed to help UAS maturing, and this section presents three reasons why that is.

1.2.1 UAS can offer new services

The U-space will allow the safe use of multiple UAS at the same time. This capability will make different services arise that could not function until now due to the lack of regulation and safety in the use of UAS. Without a shared group of rules and guidelines for these kinds of systems to follow, multiple drones offering different services would result in mayhem.

Some services are, for example, infrastructure inspections to railway tracks, dams, or power grids; agricultural uses to water and plant seeds, control crops, inspect the state of plants; services like autonomous package delivering from online shopping; surveillance, population management or even first aid. All these services can be made more autonomous and faster with the help of automated air transportation, especially in first aid services, which can make a difference in many difficult situations.

1.2.2 UAS market's growth potential

It is expected for the aforementioned services to create a lot of demand and, therefore, money. SESAR anticipates a demand evaluated in several billion EURO in the following decades. For these services to work, there will be the need to develop a human support infrastructure behind them, creating many jobs. SESAR also anticipates that by 2050 there will be a creation of more than 400.000 highly skilled jobs [27].

The increased demand and need for more jobs mean much growth for everyone involved, which motivates many people.

1.2.3 Developing the potential of UAS

As of today, there are not a lot of laws and guidelines for the usage of UAS. One of the fundamental blocks for their use in our everyday lives is a well-regulated and supervised environment. With a properly designed U-Space environment, this can be achieved.

Ultimately, and thanks to a properly regulated U-Space infrastructure, UAS-assisted technologies will reach more people, fostering the creation of new products, services and application scenarios. A similar case happened with the internet; all the ideas that derived from human creativity and the merging of technologies allowed for some incredible projects and services to come alive. The same thing will happen with air transportation, and the U-space will help accelerate that process.

1.3 Contributions

The work done in the context of the thesis will contribute to the BUBBLES' development by designing and developing the communication system proof of concept, creating a solution of at least Technology Readiness Level (TRL) 3. For BUBBLES' separation management

to work, UAS flight information needs to be consumed. Consumption of information must be done via a communication infrastructure capable of handling all the performance and safety requirements to assist drones in nominal and non-nominal conditions.

A testing model needs to be developed to assess the system's capabilities under various conditions to ensure that the communication system reaches the necessary standards. The various conditions will be used to simulate all types of environments that the separation service will need to handle in the real world.

This thesis presents the communication system, the test environment, and the steps that lead to its creation. For BUBBLES, the test environment is as important as the communication system itself because it will then use the communication solution created to help validate the Separation Management Service.

1.4 Document structure

This document is divided into eight sections. This section was an introduction to the context of the project, drone usage and its problems, projects trying to tackle said problems, their motivations, and the contributions of this work for the overall project.

Chapter 2 presents the State-of-the-Art and the research done to identify current or recently developed topics in areas related to the work of this thesis. In this case, it will be about drone communications, drone simulation, and relevant methodologies and projects within the concept of U-Space.

Chapter 3 will detail how the development of the communication service took place. First, the methodology chosen for the development of this system will be presented, followed by the tasks and the comparison of planned and actual work, and finally, the threshold of success considered for this project.

Chapter 4 presents the requirements for the project. First, a brief explanation of BUBBLES is presented, followed by a methodology to rank requirements, and finally, both functional and non-functional requirements.

Chapter 5 introduces the proposed communication architecture where, with the information on the functional and non-functional requirements, the architectural drivers will be described. Finally, with the architectural drivers and the information gathered in the State-of-the-Art section, the proposed architecture is presented.

Chapter 6 documents the development of the communication system. It will explain what was developed, what implementations were picked, how the code was organized and how the system was configured.

Chapter 7 covers the tests that were undertaken on the proof-of-concept communication platform prototype. The communication system needs to be tested under various conditions and workloads in order to validate the solution. The section will explain the test environment and the different test scenarios. In the end, the result of said tests are analysed.

Chapter 8 gathers all the main conclusions of this work and presents future work to mature this solution. It also discusses how well the objective was achieved, the deviations and mishaps of the work done, and how well the service does in the bigger scheme.

This page is intentionally left blank.

Chapter 2

State-of-the-art

This chapter is dedicated to researching the concepts, techniques, and technologies relevant to to the scope of the work undertaken in this thesis.

It will start by presenting sine of the application layer protocols most commonly used in the Internet of Things (IoT) and drones. IoT and drone ecosystems share some characteristics, and since drone communications systems are still a recent concept, the gap is filled with research on IoT communication systems.

After that, some physical layer technologies are presented to create network profiles relevant for testing purposes. Followed by drone simulation software, and, finally, the methodologies and projects relevant to this work and U-Space.

2.1 Application Layer protocols in IoT

The application layer specifies the communication protocols and interfaces used by hosts in the communications network, defining the messaging capabilities of applications. This designation is inherited from both the OSI and TCP/IP layered protocol specifications, where the upper layers correspond to higher levels of abstraction regarding the lower levels, increasingly closer to the physical medium as we move downwards.

IoT communication systems are deployed everywhere and are characterized by many clients communicating in low-power machines sending information periodically. This characterization can be used to describe the drone system that is going to be built. Because of this, when picking protocols to research, their maturity and widespread use in the context of IoT was considered, as well as their licensing (i.e., if they are open-source or not) and related academic research.

The application protocols selected for evaluation were the Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT), and Apache Kafka. In the following subsections, the different application protocols are presented. Each subsection will describe the protocol's architecture, message format, security, and quality of service options. The final subsection (Protocol Comparison) will compare the protocols and show other research done on the protocols.

2.1.1 AMQP

AMQP [31] is an open standard application layer protocol for message-oriented middleware and describes itself as an internet protocol for business messaging. It is an asynchronous message queueing protocol originally designed for financial transaction processing. For the protocol to work, it assumes a reliable connection protocol like TCP/IP.

The last AMQP version is v1.0 and is the only one standardized by OASIS. This version is radically different from the earlier versions [32] and focuses on how the data is transferred on the wire; as such does not define any requirements on broker internals and is entirely broker-model agnostic. Therefore, due to the context of this thesis, the latest version that describes the broker's functioning will be chosen (v0.9.1).

Architecture

The AMQP specification [31] defines two layers. A functional layer defines the set of commands that do the needed work on behalf of the applications; and a transport layer that carries the information from application to server and back.

AMQP's functional layer has two main entities, as illustrated in Figure 2.1. The Exchange and the Queue [31].

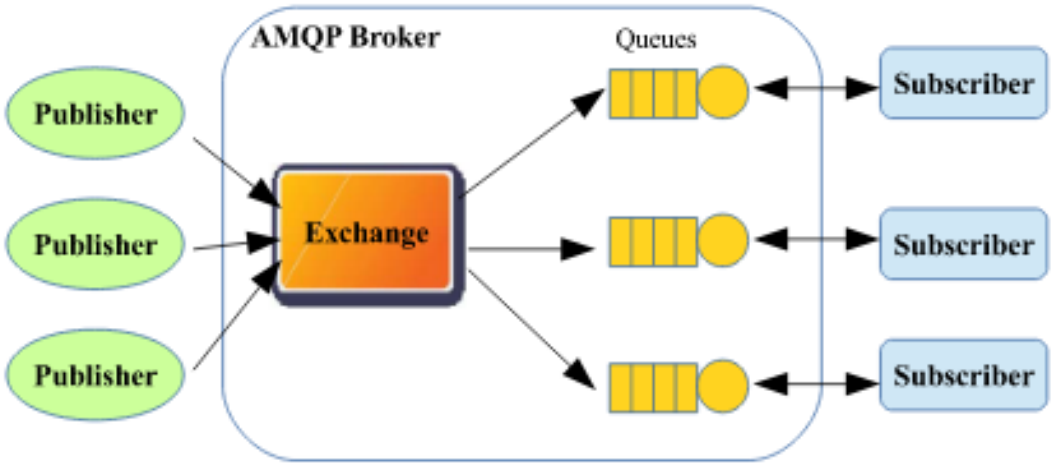

Figure 2.1: AMQP's Architecture (from [1]).

Messages are sent to what AMQP designates by *exchanges* instead of sending them to queues directly. Exchanges are matching and routing engines. After receiving the message, exchanges then send the message to queues based on pre-defined criteria. If the message queue matches the criteria, it shall receive a copy of the message. These criteria are called "bindings."

A binding is a relationship between a message queue and an exchange. The binding specifies the routing arguments that tell the exchange which message queue should get the message.

Bindings can be defined by a routing key, a routing pattern, or a table of arguments. The routing key is a virtual address. A routing pattern is a way of expressing multiple routing keys (e.g., routing key: thesis.chapter.state-of-the-art, routing pattern: thesis.chapter.*). Finally, a table of arguments is a set of names (for headers) that a message should have,

and optionally what values they should hold.

The queue will store the message in memory or on disk and send it to subscribed consumers. Thus, message queues are message storage and distribution entities. Each message queue is entirely independent of one another. This way of communicating allows point-to-point communication, and it also supports the publish-subscribe communications model.

In AMQP, a producer does not need to maintain any information or state about the messages' consumers. Likewise, a consumer does not need to maintain any information or state about the message's producers. The exchange decides to which queue messages are routed instead of the application, which reduces complexity on the application side. The documentation expresses two mandatory and two advised types of exchanges.

The Direct Exchange Type (Mandatory). In this case, the message queue binds to the exchange utilizing a routing key. The message queue shall receive the message if the message's routing key matches the queue's routing key.

The Fanout Exchange Type (Mandatory). In this case, the message queue binds with no arguments and will receive every message unconditionally.

The Topic Exchange Type (Advised). In this case, the message queue binds using a routing pattern. The message queue shall receive the message if the message's routing key matches the pattern.

The Headers Exchange Type (Advised). In this case, the message queue binds using a table of arguments. If the message's headers and their values (if configured) match the ones in the binding, the message queue will receive the message.

Message format

In the transport layer, its information is organized into frames.

As shown in Table 2.1, a typical AMQP frame has five sections, type, channel, size, payload, and frame-end.

<table>
<tr><th>bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>Byte 1</td><td>type</td><td colspan="3">channel</td><td colspan="4">size</td><td rowspan="2"></td></tr>
<tr><td>Byte 2</td><td colspan="7" rowspan="2">Payload</td></tr>
<tr><td>Byte N</td><td>frame-end</td></tr>
</table>

Table 2.1: AMQP's Frame Format (adapted from [3]).

The type section is to specify the type of data that is going in the payload. There are four possible types which are: Method, Header, Body, and Heartbeat.

When the type is Method, it carries an Remote Procedure Call (RPC) request or response. AMQP uses an RPC pattern for nearly all kinds of communications between the broker and the clients. For example, when a client is publishing a message, the application calls the method Basic.Publish, and this message is carried in a Method frame that will tell the broker that a client will publish a message.

When the type is Header, it contains the properties of another frame. Certain specific methods carry content (e.g., the message to be published), and the content header frame is used to send the properties of this content. For example, this frame may have the content

type of a message that will be published and a timestamp.

The third type is the Body type, which is the frame with the actual content of a message. It can be split into multiple frames if the message's size is too big (131 KB is the default frame size limit).

The last type is the Heartbeat type. This one is used to confirm that a given client is still alive. If the broker sends a heartbeat to a client and does not respond quickly, the client will be disconnected, as it is considered dead.

The channel section is to specify the communication channel. This value is 0 for global frames to the connection and 1-65535 for frames that refer to specific channels. AMQP allows channel multiplexing, which is the use of the same socket for various threads. With each thread dealing with one channel.

The size section is to specify the size of the payload. This section is used to detect framing errors caused by incorrect clients or bad implementations.

The payload section holds the message data and starts at the eighth bit of the first byte. It begins there because, in the end, in the last bit of 1 + N Bytes, where N is the payload size, is the frame-end section, which is a single bit to determine the end of the frame.

Security

Both AMQP versions (v0.9.1 and v1.0.0) can be used simultaneously [33]. The latest one can extend certain security functionalities like Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL) [34] to the earlier one. SASL is a framework that allows for authentication based on different mechanisms. TLS uses certificates to verify the server's identity, and optionally, the client's and provides a two-way encrypted channel between the two.

Generally, AMQP implementations use SASL to provide authentication and uses TLS to secure the tunnel. Note that there is no mandatory security requirement in the AMQP v0.9.1 specification [31]. Because of that, the security provided depends on the implementation.

Quality of Service

Generally, AMQP supports reliable communication via message delivery guarantee primitives, including at-most-once and at-least-once. Exactly once delivery is not commonly used (e.g., IBM's implementation of AMQP [35] only uses at-most-once and at-least-once). Note that there is no mandatory quality of service requirements in the AMQP v0.9.1 specification [31]. Because of that, the quality of service provided depends on the implementation.

2.1.2 MQTT

MQTT is a lightweight Machine to Machine (M2M) publish-subscribe messaging protocol. It runs over TCP/IP or other network protocols that provide ordered, lossless and bidirectional connections. Due to the nature of the target devices (embedded devices and networks), MQTT aims to offer communications with low overhead.

This protocol was created by Andy Stanford-Clark and Arlen Nipper in 1999. At the time, it was part of IBM's MQ Series message queueing product line. In 2013, IBM submitted MQTT v3.1 to the OASIS specification. MQTT has been a standardized protocol ever since. The latest version, and the one researched, is version 5.0.

Architecture

MQTT [3] gets inspiration from the old client/server model. Every device is a client that connects to the server, known as the broker.

Clients will send (publish) messages to an address known as a topic. Clients can subscribe to a topic to receive the messages sent to it. A name represents each topic, and it exists within brokers. Every message received in a topic is a discrete chunk of data, opaque to the broker.

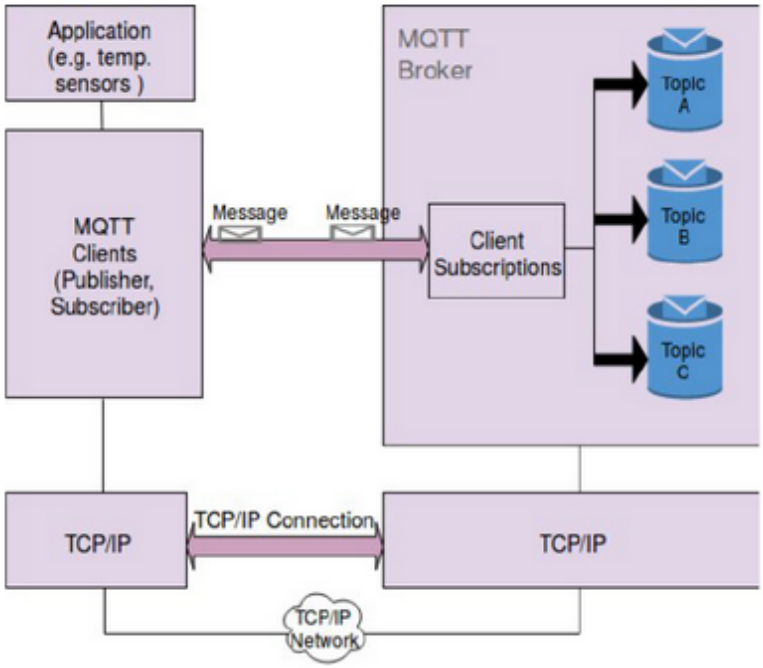

Figure 2.2: MQTT's Architecture (from [2]).

A Client can be a Publisher or Subscriber, and it always establishes the network connection to the Server (Broker). It can do the following things: Publish messages into topics; subscribe to topics; unsubscribe to extract from the subscribed subjects, and detach from the broker.

The broker controls the distribution of information and mainly responsible for receiving all messages from publishers, filtering them, deciding who is interested in it, and then sending the messages to all interested clients. It can do the following things: Accept Client requests; receive Published messages by clients; process different requests; and unsubscribe from Clients.

MQTT also supports the request-response pattern asynchronously. "Response topic" is an optional parameter that represents the topic(s) on which the response(s) from the receiver(s) of the message is expected. In addition, to identify which request is the response

received related to, the sender must set the message's correlation data parameter, and the response will come with that same value.

Message format

Table 2.2 illustrates the message format of a packet containing a message to publish.

The first byte is the Fixed Header and is divided into two sections. Bits zero to 3 are flags specific to each message type, and bits 4 to 7 are the message type. The first four bits are only used in publish packets, with the fields retain, QoS and DUP.

If the packet has the retain field set to 1, the server must store the message and its Quality of Service (QoS) to be delivered to future subscribers of the same topic. The QoS field is used to express the QoS option of the packet. If set to 0 means at-most-once delivery, 1 means at-least-once delivery, and 2 means exactly-once delivery. The DUP field and is used to indicate duplication. Every message that is resent (fails the first try) needs to set this field to 1.

bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Byte 1	Retain	QoS		DUP	Message type			
Byte 2	Remaining Length							
Byte 3	Variable Header							
Byte N								
Byte N+1	Payload							
Byte M								

Table 2.2: MQTT's Message Format (adapted from [3]).

The last field in the Fixed Header is the message type. MQTT has fifteen types to choose from and is what allows the QoS options it offers.

- CONNECT - The client request a connection to a Server.
- CONNACK - Acknowledge connection request.
- PUBLISH - Publish message.
- PUBBACK - Publish acknowledgment (QoS 1).
- PUBREC - Publish received (QoS 2 publish received, part 1).
- PUBREL - Publish release (QoS 2 publish received, part 2).
- PUBCOMP - Publish complete (QoS 2 publish received, part 3).
- SUBSCRIBE - Subscribe to topics.
- SUBACK - Subscribe acknowledgment.
- UNSUBSCRIBE - Unsubscribe from topics.
- UNSUBACK - Unsubscribe acknowledgment.
- PINGREQ - PING request.

- PINGRESP - PING response.
- DISCONNECT - Disconnect notification.
- AUTH - Authentication exchange.

The Remaining Length Section starts at byte 2. It is a Variable Byte Integer that represents the number of bytes remaining within the current message, which is the data in the Variable Header and the Payload.

The information in the Variable Header depends on the packet type. Still, it usually has the packet identifier and a set of properties that can encode information like correlation data, payload format indicator, and topic name. Finally, the payload contains the actual data of the message, which may be nothing depending on the packet type.

Security

In MQTT, the most basic type of client authentication is via the CONNECT packet, which may include username and password fields. The packet type AUTH allows for what the specification [3] calls Enhanced Authentication. It allows for Applications to configure challenge/response style authentications. This can be used with SASL to offer, for example, OAuth or LDAP authentication.

TLS can be used, which will authenticate the server, and optionally the client, by means of certificates. A series of cipher suites can be used in TLS but there is no mandatory security requirement in the MQTT specification.

Quality of Service

MQTT has 3 QoS levels.

The first level guarantees at-most-once delivery. This means that the subscriber may receive the message 0 or 1 time. The receiver accepts ownership over the message when it receives the packet. Figure 2.3 illustrates the flow of communication for QoS 0.

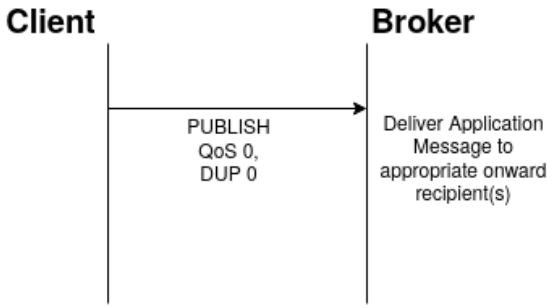

Figure 2.3: MQTT's QoS 0 flow (adapted from [3]).

The second level guarantees at-least-once delivery. This means that the subscriber may receive the message one or more times. The receiver accepts ownership over the message when the sender receives the last PUBBACK. Figure 2.4 illustrates the flow of communication for QoS 1.

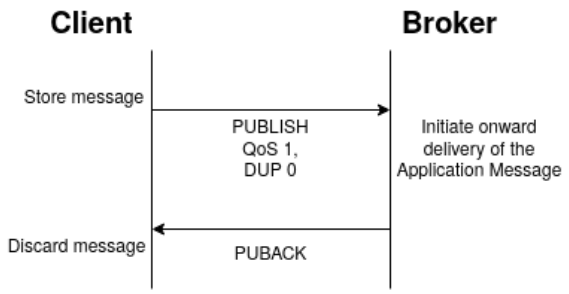

Figure 2.4: MQTT's QoS 1 flow (adapted from [3]).

The third level guarantees exactly-once delivery. This means that the subscriber receives the message one and only one time. The receiver accepts ownership over the message when the sender receives the PUBREC. In order to achieve this highest level, there is an increased overhead in communications. Figure 2.5 illustrates the flow of communication for QoS 2.

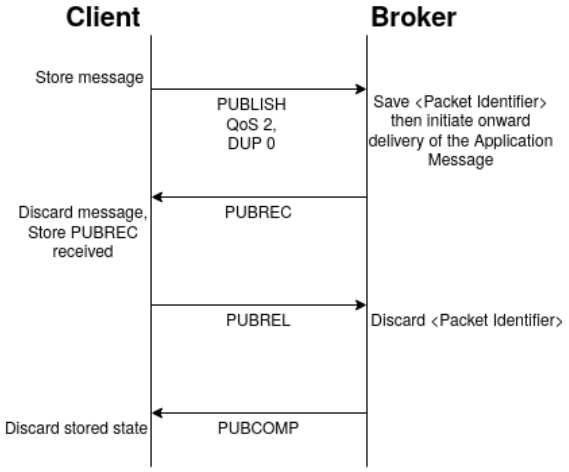

Figure 2.5: MQTT's QoS 2 flow (adapted from [3]).

Messages that have the retain flag set to 1 will be stored on the disk. In case of an error occurs and the receiver is unable to get the packet, a delivery retry will proceed. When a client initiates a connection, if it is requested without the clean state set to 1, and an earlier session is present, the messages stored that were not delivered will be sent again. MQTT supports message ordering for QoS 1 and 2, although in QoS 1 duplicates may arrive after they should.

Finally, MQTT allows clients to configure one last message in case of imminent failure. When connecting to the broker, clients can specify the last will (typical MQTT message with a topic), and the broker will send it when it detects that the client has died.

MQTT-SN

It is a failed attempt to build an MQTT for sensor networks, where devices have low power battery, limited payload size, and not always on. The main difference to the original protocol is the reduced size of the message payload, and instead of requiring a TCP connection,

it uses UDP as the transport protocol.

It never got much use, and there is not much research about it. For those reasons, it was not taken into consideration when selecting the protocols.

2.1.3 Kafka

Apache Kafka [36] is an open-source message-broker middleware with additional streaming capabilities. This means that it can publish and subscribe to data, can store the data in question, and can process data before distributing it to subscribers. It only functions over TCP/IP.

Kafka was designed with performance and persistence in mind. Apache claims that Kafka can "act as a unified platform for handling all the real-time data feeds a large company might have" [36].

Kafka was originally developed by LinkedIn in 2011. The protocol "graduated" from Apache Incubator on 23rd October of 2012 [37] and has since then been maintained by the Apache Software Foundation. The latest version, and the one researched, is version 2.8.

Architecture

Kafka shares the same principles as other message-oriented middleware. Producers push messages to a queue while Subscribers retrieve those messages. Figure 2.7 shows the topic view architecture for Kafka.

A topic can be viewed as the common queue in message-oriented architectures. But instead of producers sending messages to a topic, they send messages to a topic's partition. A partition is a subdivision of a topic, and brokers have the responsibility of zero to N partitions, being N the number of partitions a topic has.

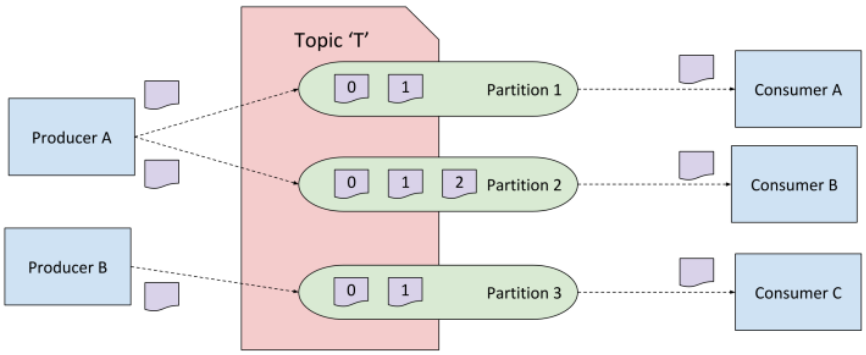

Figure 2.6: Kafka's High Level Architecture (from [4]).

Kafka's focus on performance and persistence shows in how brokers store their messages. Instead of writing information to an in-memory cache, controlled by Kafka, and flushing it to disk when needed, all information is written to a persistent log on the filesystem, without necessarily flushing it to disk.

The persistent log on the filesystem is the kernel's pagecache, which is a transparent cache for pages (contents of files) from storage devices like a hard disk. Pages with new

information (dirty pages) relative to the disk are written to the disk periodically. Messages are deleted on two occasions. Applications configure a time limit for information, from deleting it right away to keep it forever or configure a size limit for the persisted files.

If data was stored in a cache controlled by the application and the program crashes, all the cached data would be lost. By storing it on the filesystem memory, program crashes do not affect pagecache's information. In addition, since writes are made into the pagecache system, reads can also be made from it, which contributes greatly to its performance.

When a message is stored in a partition, it is given a unique, incremental integer identifier called offset. The message can only be sent to subscribers when it is committed (stored) because it is by using this offset that Consumers fetch data. As a consequence, messages are committed and delivered in the order they were sent, although if messages consumed are coming from different partitions, Kafka does not guarantee their order. This way, Consumers can also reproduce past messages, if they were not yet deleted, using an earlier offset.

Figure 2.7 shows Kafka's high-level architecture. Clients that want to publish or consume data can only communicate with the partition's leader (green partition in the image below). Other brokers with the same partition are called followers. The leader is responsible for balancing communication loads, distributing written messages to followers, and keeping track of in-sync replicas (followers fully caught up with the leader). Messages are only considered as committed if all in-sync replicas have received a copy of the message.

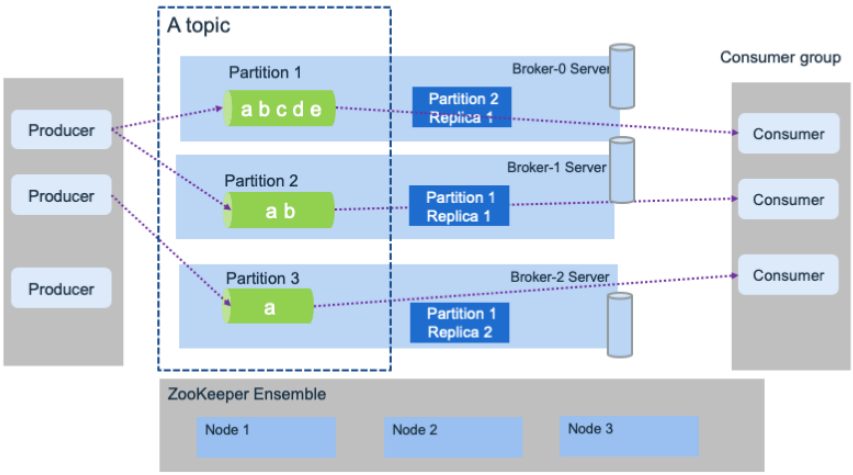

Figure 2.7: Kafka's Architecture (from [5]).

In Kafka, the notion of a consumer in Kafka is generalized to be a group of co-operating processes running as a cluster. Because of that, a consumer is a Consumer Group with just one consumer. Consumers should belong to the same group if they do the same task on the same topics and partitions (e.g., process information of specific sensors).

Each message in the topic is delivered to one consumer in each of these consumer groups. The last read message offset for a partition is saved, by the brokers, on a per-consumer group basis. As such, different consumer groups can process information at their own pace.

From Figure 2.7 we can see that Kafka by itself is not sufficient. It requires the help of Zookeeper. Three configurations are essential to set up a Kafka broker, and they are the identifier of the broker (broker.id), where to write the output to (log.dirs), and where to

connect to ZooKeeper (zookeeper.connect).

Apache has stated that Kafka is walking in the direction of being independent of ZooKeeper [38]. At the time of this thesis, it is not there yet.

Currently, Zookeeper keeps the information about cluster configurations and membership. The brokers send periodic heartbeat signals to ZooKeeper. If no heartbeat signal arrives, then the broker is considered dead and removed from the cluster. If the one considered dead is the leader of a partition, ZooKeeper will aid in choosing a new leader from the remaining brokers. ZooKeeper is also responsible for controller broker election (responsible for maintaining the leader/follower relationship of all partitions), topic configuration (list of topics, number of partitions, and so on), Access Control Lists (ACLs) (lists that specify authorization for clients) and Quotas (how much data is a client is allowed to write/read).

Message format

Kafka defines two types of messages. Both are key-value pairs with some metadata associated. First are the Request and Response messages. These are used between the client and the broker to identify actions or information that the client will perform/needs (e.g., fetch information or ask which is the address of a partition's leader). Table 2.3 shows the most basic Request format.

bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Byte 1	Request Api Key							
Byte 2								
Byte 3	Request Api Version							
Byte 4								
Byte 5	Correlation ID							
Byte 9								
Byte 10	Rest of the message size							
Byte 14								

Table 2.3: Kafka's Request Header format (adapted from [15]).

The Header is composed of the request_api_key, request_api_version, and correlation_id. The first one is to identify what type of request is the current packet. Kafka currently defines 61 different types of requests, more information in [15]. The second one is to inform about the Kafka's version being used, and the last is used to match responses from the broker to a previous request by the client. After the Header, all requests start with the 4-byte integer, which specifies the size of the rest of the message.

The other type of message is Records. These Records are used to contain the data published and consumed by applications. They exist within some Requests, like the Produce and Fetch Requests.

In table 2.4, we can see that a Record contains eight fields. Varint and varlong both describe a type of encoding numbers that allows for an unknown number of bytes to be used. Both varint and varlong start with the first set to 1 if more bytes are needed to write the number and start with the bit with zero if it is the last octet. The difference between the two is that var int has a maximum of 5 bytes and varlong a maximum of 10 bytes.

- length - Size of the Record.

bit	0	1	2	3	4	5	6	7
Byte 1	1	length (varint)						
Byte L	0	rest of length						
Byte L + 1	attributes							
Byte L + 2	1	timestampDelta (varlong)						
Byte L2	0	rest of timestampDelta						
Byte L2 + 1	1	offsetDelta (varint)						
Byte L3	0	rest of offsetDelta						
Byte L3 + 1	1	keyLength (varint)						
Byte L4	0	rest of keyLength						
Byte L4 + 1 + keyLength	key							
Byte +1	1	valueLen (varint)						
Byte L5	0	rest of valueLen						
Byte L5 + 1 + valueLen	value							
Byte H Length	Headers[]							

Table 2.4: Kafka's Record Format (adapted from [15]).

bit	0	1	2	3	4	5	6	7
Byte 1	1	keyLength (varint)						
Byte L1	0	rest of keyLength						
Byte L1 + 1 + keyLength	key							
Byte +1	1	valueLen (varint)						
Byte L2	0	rest of valueLen						
Byte L2 + 1 + valueLen	value							

Table 2.5: Kafka's Record Header Format (adapted from [15]).

- attributes - Specify attributes for the Record. Not currently in use.
- timestampDelta - Value to add to firstTimestamp to get Record's timestamp.
- offsetDelta - Value to add to firstOffset to get Record's offset, and to firstSequence to get Record's sequence.
- keyLength - The size of the key.
- key - Key information
- valueLen - The size of the value.
- value - Value data.

At the end of the Record, there can be a list of Headers to add additional properties to the message. Each header in the list, as shown in Table 2.5, is just a key-pair value.

A RecordBatch is the structure used for on-disk storage and may contain one or more records. Table 2.6 shows the RecordBatch format.

bit	0	1	2	3	4	5	6	7
Byte 1	baseOffset							
Byte 8								
Byte 9	batchLength							
Byte 12								
Byte 13	partitionLeaderEpoch							
Byte 16								
Byte 17	magic							
Byte 18	CRC							
Byte 21								
Byte 22	compression			timestampType	isTransactional	isControlBatch	unused	
Byte 23	unused							
Byte 24	lastOffsetDelta							
Byte 27								
Byte 28	firstTimestamp							
Byte 35								
Byte 36	maxTimestamp							
Byte 43								
Byte 44	producerId							
Byte 51								
Byte 52	producerEpoch							
Byte 53								
Byte 54	baseSequence							
Byte 57								
Byte 58	records[]							
Byte 58 + N								

Table 2.6: Kafka's RecordBatch Format (adapted from [15]).

- firstOffset - Specifies the offset of the first Record in the Recordbatch.
- batchLength - Specifies the length of the RecordBatch.
- PartitionLeaderEpoch - For broker's internal use. The broker sets it upon receipt of a Publish request to ensure no loss of data.
- Magic - Specifies the version id. It currently needs to be set to 2 (after version 0.11)
- CRC - The CRC is the CRC32 of the remainder of the message bytes. CRC is used to check the integrity of the message on the broker and consumer.
- Attributes - Specifies the compression codec used, the timestamp time (CreateTime vs. LogAppendTime), if the RecordBatch is part of a transaction and if the batch includes a control message.
- lastOffsetDelta - The offset of the last message in the RecordBatch. Used to confirm correct behavior within batch compression.
- firstTimestamp - The timestamp of the first Record in the batch.

- maxTimestamp - The timestamp of the last Record in the batch.
- producerId - This is the broker assigned producerId.
- producerEpoch - This is the broker assigned producerEpoch.
- firstSequence - Each message is assigned a sequence by the Publisher, which the broker uses to deduplicate messages. Publishers who want to use idempotent message delivery and transactions must set this field. The sequence number for each Record in the RecordBatch is its OffsetDelta + FirstSequence.

Security

Kafka has three entities that need authentication: the client, a publisher or a consumer, the brokers, and the Zookeeper instances.

The client can authenticate two ways, with TLS or SASL. TLS authenticates the user using certificates and encrypts the channel. Kafka supports the following SASL mechanisms: GGSAPI, PLAIN, SCRAM-SHA-256, SCRAM-SHA-512, OAUTHBEARER.

In addition to authentication, Kafka supports the use of ACLs to limit the authorization of the client to do specific tasks, like write or read a topic. One can even specify from what IP address the client is supposed to be communicating.

The brokers and Zookeeper instances authenticate one another by the use of TLS. In addition, brokers can also authenticate via SASL.

Quality of Service

Kafka can be configured to achieve the three basic QoS types: at-most-once, at-least-once, and exactly-once. The concept for what exactly-once means depends if the point of view is from the Producer or the Consumer. From the producer's point of view, exactly once means that the message published arrived at the broker and was only logged one time. In the default scenario, at least once, if the broker receives a message and logs it, then crashes and does not send the ACK to the producer, the producer will resend the message. This will result in a duplicate in the logs.

To support exactly once, Kafka has the notion of idempotent requests. The producer must set a sequence number for each message. Thus, the broker can still receive multiple messages, but it can ensure the message is not duplicated into the logs by checking if the sequence number is already in the logs. The producer can configure how long to wait for ACKs. The options are no waiting (at-most-once), wait just for the leader, or wait for every in-sync replica.

From the consuming side, exactly once means that the message was consumed and processed by the consumers only one time. Consumer configuration affects the type of guarantee. If the consumers start by receiving a message, then updating the offset and end with processing it, if it fails during the process part, Kafka will assume it is processed, even though it is not. This is the configuration for at-most-once communications. For at least once, the consumer must process the information first and then update the offset. In case of failure during the update offset part, the consumer will process the data again. This is the default option. It might process it one or more times.

In order to offer exactly-once consumption, the consumer must use the transactional feature, where every message is delivered, or none do. The only caveat is that consumers must publish the result of the processing to a Kafka topic. With this, the result message and the update offset message can both be in a transactional message. This will guarantee the exactly-once delivery.

2.1.4 Protocol Comparison

AMQP, MQTT, and Kafka work differently; however, all can be considered message distribution systems. Table 2.7 shows the summary of the literature review for this section.

The research column refers the literature sources that helped understand what protocols were used in the context of IoT and learn more about each protocol. Next, the performance tests column shows the literature related to testing the different protocols in terms of communication performance (e.g., their throughput, latency, as so on). Finally, the security column shows the literature related to researching and assessing the security vulnerabilities of the protocols.

Protocol	Research	Performance Tests	Security
AMQP	[1] [39] [40]	[4] [16] [41]	[42]
MQTT	[1] [39] [2]	[16] [43]	[44]
Kafka	[40] [39] [45]	[4] [41]	-

Table 2.7: Sources studying related protocols.

All three protocols run over TCP. AMQP allows both Point-to-Point and Publish-Subscribe communications using a Direct Exchange and Fanout or Topic Exchanges, respectively. MQTT allows Publish-Subscribe pattern by using topics and an asynchronous Request-Response type of communications using Response Topics. Kafka only supports the Publish-Subscribe pattern by the use of topics and partitions.

In AMQP, every packet size is 1 Byte plus the size of the payload. In MQTT, packets have a header with the size of 1 Byte, plus the variable header and payload. Kafka's message size depends on the type of request. For example, using Produce requests as the reference, messages will have 14 Bytes for a Header, plus the payload, which includes the records. Each record has its metadata associated, having at the bare minimum 8 Bytes of metadata for the record.

All three offer message persistence capabilities, but Kafka's goes the extra mile, configuring how long the information is available and saving various replicas of information between brokers in the cluster.

In terms of security, AMQP can offer both TLS and SASL if both versions of the protocol are used. So it can both authenticate the clients and the brokers, as well as encrypt their communications. [42] is an analysis of security vulnerabilities and cyber threats on AMQP. It reports various vulnerabilities that existed (in 2017) in AMQP protocol and how to exploit these weaknesses to make AMQP a susceptible protocol. It presents 17 attacks.

MQTT supports different authentication options, including TLS, SASL, and other challenges/response style authentications. It can use TLS also to encrypt the channel, but there are no mandatory requirements in the specification. [44] analyses MQTT security requirements, describes different attacks and attempts to recreate them (and successfully

does). It presents five attacks.

Kafka supports both TLS and SASL authentication for both the client and the broker and only TLS for Zookeeper instances. In contrast to AMQP and MQTT, it also supports the use of ACLs to limit clients' usage and interaction. [46] is a list of CVE vulnerabilities that Kafka has fixed. It lists all the vulnerabilities found and described by CVE related to Kafka, which may explain why no Kafka Papers were found on the subject. [46] also explains what triggers the vulnerability and what version fixes the problem.

AMQP has no quality of service requirements on the specification but typically offers both at-most-once and at-least-once. MQTT offers the three types, at-most-once, at-least-once, and exactly-once, specifying them in the documentation. Kafka also offers three QoS levels, but consumers reading exactly once is only guaranteed if consumers publish their output to Kafka.

Performance-wise, [16] compares AMQP and MQTT. It states that AMQP slightly outperforms MQTT in latency with message sizes bigger than 5000 Bytes. However, AMQP uses more CPU and memory than MQTT. Furthermore, [4] and [41] compare AMQP and Kafka, both state that Kafka outperforms AMQP in throughput. In terms of latency, [4] states that AMQP has the lower latency, [41] also states this, but only for small amounts of messages. Unfortunately, no paper comparing Kafka and MQTT was found.

Finally, from a native scalability point of view, AMQP and MQTT offer almost no scalability options. In contrast, Kafka offers excellent scalability options, like adding more brokers to a cluster, adding more partitions to a broker, or adding consumers to a consumer group. It also balances loads automatically. Load balancing and redundancy can be achieved in AMQP and MQTT using external tools, which was not considered when comparing the protocols.

Table 2.8 presents a summary of the comparison.

Property	AMQP	MQTT	Kafka
Transport Layer	TCP	TCP	TCP
Message Distribution	Point-to-Point and Publish-Subscribe	Request-Response and Publish-Subscribe	Publish-Subscribe
Message Overhead	1 Byte	1 Byte + Variable Header	14 Bytes + 8 Bytes + Additional Metadata
Message Durability	Yes	Yes	Yes, and for how long
Security	TLS & SASL (with v.1.0)	TLS & SASL & Username + password	TLS & SASL & ACLs
QoS Levels	Usually 2	3	3
Scalable	With external tools	With external tools	Yes

Table 2.8: Protocols comparison.

2.2 Application Layer protocols in drones

UAS usage in complex communication systems is still something recent, and as such, there is not much research on the subject. Drones share many similarities with IoT devices, in

the sense that are often based on embedded systems with not much power of bandwidth available, but they are also critical systems, where low latency is crucial for the drone's functioning. So it is not any protocol that will fulfill the drone's needs. For the protocols, their use in vehicle-related scenarios was considered, as well as their licensing and research, and the application protocols picked are as follows: MQTT, Data Distribution Service (DDS), and MAVLink.

Similar to the earlier section, in the following subsections, the different application protocols are presented. Each subsection will describe the protocol's architecture, message format, security, and quality of service options. After that, some honorable mentions are presented. Although some additional protocols appeared during the research, they are not ready to be used in the proposed solution, but a short description is presented as they might mature into a feasible solution in the future. The final subsection (Protocol Comparison) will compare the three protocols and show other research.

2.2.1 MQTT

MQTT has seen some use recently in Vehicle to Everything (V2X) communications. Some examples are [47], [48], and [49]. In addition, its use in the context of UAS is not unheard of, for example [50], [51], and [52]. For those reasons, it was considered as a candidate for drone communications.

For more information about the protocol, refer to subsection 2.1.2.

2.2.2 DDS

DDS is a publish-subscribe protocol for soft real-time M2M communications standardized by the Object Management Group (OMG). Its development started in 2001 with Real-Time Innovations (RTI), and in 2004 OMG published its first version.

DDS has a broker-less architecture and uses multicast to provide reliable and efficient communications on UDP and TCP networks. The latest version, and the one researched, is version 1.4.

Architecture

DDS architecture defines two layers: Data-Centric Publish-Subscribe (DCPS) and Data Local Reconstruction Layer(DLRL). DCPS is responsible for delivering the information to the proper recipients. DLRL, on the other hand, is an optional layer and serves as the object-model interface to the DCPS functionalities. For the context of this thesis, only the DCPS was taken into consideration. The DDS communication model [6] consists of five components (see Figure 2.8): The Publisher, Subscriber, DataWriter, DataReader, and Topic.

The Publisher and Subscriber components handle messages. A Publisher is an object responsible for data distribution. A Subscriber is an object responsible for receiving the data published and making it available for the application it belongs. The Subscriber object chooses what messages to make available according to its own QoS.

A DataWriter and DataReader act as typed accessors to both the Publisher and Subscriber, respectively. In order to send data, the Publisher must use a DataWriter. To read the

information, the Subscriber must use a DataReader. Both the Data Writer and Reader can only bind to one type of data. As such, Publications are defined by the association of data-writers to Publishers, and Subscriptions are defined by the association of data-readers with Subscribers.

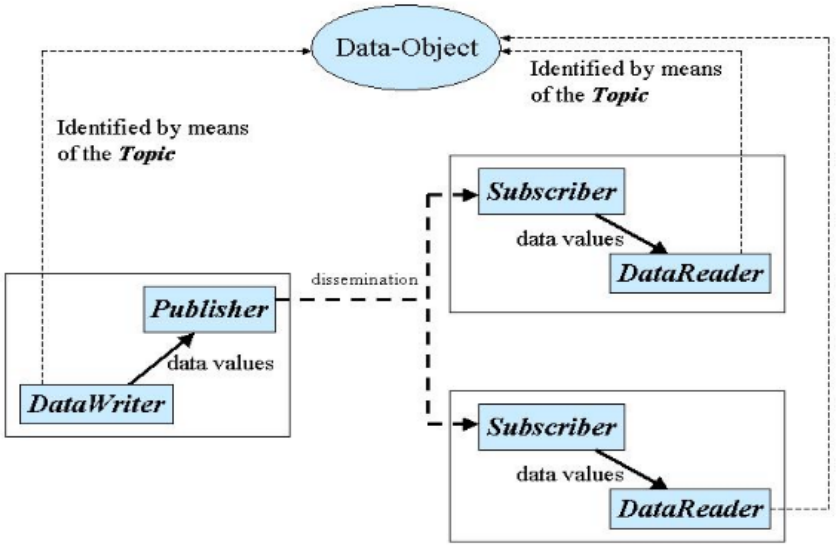

Figure 2.8: DDS's Architecture (from [6]).

The subscriptions need an unambiguous way to refer to publications. Topics are meant to fulfill this purpose. Topic objects conceptually fit between publications and subscriptions. It associates a name (unique in the domain), a data type, and QoS related to the data itself. A domain is what DDS defines as the communication network plane. Only the Publishers and the Subscribers attached to the same domain may interact. Subscribers and Publishers belong to a domain participant, representing the local membership of the application in a domain.

Because DDS has a broker-less architecture, they need a way of finding its peers within the domain. Discovery is what allows the different nodes (DDS objects) to find each other [53]. Each DomainParticipant starts with an initial list of peers. This database will be refreshed periodically when the Simple Discovery Protocol (SDP) is performed. SDP defines the process for discovery and is divided into two phases: Simple Participant Discovery and Simple Endpoint Discovery.

In the first phase, DomainParticipants learn about each other by sending broadcasts. These declaration messages include a globally unique identifier (within the domain), transport locators (addresses and port numbers), and QoS. The declaration messages are also used to communicate changes to the DomainParticipant's QoS. When DomainParticipants receive this message, they see if there is a match based on the Domain Id and Domain Tag. If not, the DomainParticipant is discarded.

In the following phase, Simple Endpoint Discovery, applications that found each other will share information on their DataWriters and DataReaders to see if they match based on topic, data type, and compatible QoS levels. If both sides agree to communicate, a new DomainParticipant is discovered.

Message format

DDS does not define its wire protocol. Instead, it uses the Real-Time Publish-Subscribe (RTPS) protocol as its wire protocol. The RTPS message is composed of a Header followed by a variable number of sub-messages (see Table 2.9).

bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Byte 1	Header							
Byte 16								
Byte 17	SubMessage							
... ...	... ...							
Byte 17+N	SubMessage (if any)							

Table 2.9: DDS's Message Format (adapted from [16]).

The header is 16 bytes long, composed of four different sections (see Table 2.10). First, it is the Protocol field that is 4 bytes long. This value needs to be set to "RTPS." The second value is the Version field that is 2 bytes long. It specifies the RTPS version. The third field is the VenderId, which is also 2 bytes long. Each implementation of DDS will get this value from OMG. The last field is the GuidPrefix, which is 8 bytes long. Specifies a default prefix so that each sub-message only needs the entity's id to recreate the globally unique identifier.

bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Byte 1	R							
Byte 2	T							
Byte 3	P							
Byte 4	S							
Byte 5	Version							
Byte 6								
Byte 7	Vendor ID							
Byte 8								
Byte 9	GuidPrefix							
Byte 16								

Table 2.10: DDS's Header Format (adapted from [17]).

All sub-messages start with a header. This header has three fields, the first one is submessageID. This one states the kind of sub-message. The second field is "flags." The first bit is the only one that is shared by all sub-messages. It is used to identify the endianness used to encapsulate the sub-message. The other 7 bits are dependant on the type of sub-message. And finally, the sub-message length. It indicates the length of the first sub-message. If the value is zero, it only has one message that is bigger than the value this field can hold (64k).

There are 12 sub-messages ids one for each type. They are as follows:

- Data - Contains information regarding the value of an application Data-object.
- DataFrag - Equivalent to Data, but only contains a part of the new value. Allows data to be transmitted as multiple fragments to overcome transport message size limitations.

bit	0	1	2	3	4	5	6	7
Byte 1	submessageID							
Byte 2	flags							
Byte 3	octetsToNextHeader							
Byte 4								
Byte 5	Submessage Payload							
Byte N								

Table 2.11: DDS's Sub-message Format (adapted from [16]).

- Heartbeat - Describes the information that is available in a Writer.
- HeartbeatFrag - For fragmented data, describes what fragments are available in a Writer.
- Gap - From Writer to Reader to indicate that a range of sequence number is no longer relevant to Readers.
- AckNack - Reader informs the Writer about the sequence number it has received and which ones are missing.
- NackFrag - Reader informs the Writer about specific fragment numbers it is still missing (differs from AckNack because AckNack is both positive and negative fragments).
- InfoSource - It modifies the source of the sub-messages that follow.
- InfoDestination - From Writer to Reader to modify the GuidPrefix used to interpret the entity ids appearing in the following sub-messages
- InfoReply - From Reader to Writer to specify where to send the reply to a sub-message following the same message.
- InfoTimestamp - Used to send a timestamp that applies to the sub-messages that follow in the same message.
- Pad - Serves to meet any desired memory alignment requirements.

Security

DDS defines its security model as a plugin-based model [54]. This model works with five Service Plugin Interfaces (SPIs) to provide information assurance. The five SPIs are as follows: Authentication, AccessControl, Cryptographic, Logging, and DataTagging. Users can either build their SPIs or use the DDS built-in SPIs.

Without the Authentication plugin, any DomainParticipant is allowed to join a Domain. During discovery, if the DomainParticipant has the authentication plugin enabled, it will only register other DomainParticipants with the same plugin enabled. Both DomainParticipants can use the authentication process to authenticate each other. Often a shared secret is also derived from a successful authentication that the cryptographic plugin can use.

DDS built-in Authentication Plugin, DDS:Auth:PKI-DH [54]. Implements authentication by using a trusted Certificate Authority (CA).

The AccessControl Plugin enforces policy decisions on what operations that an authenticated user can perform. For example, whether to allow unauthenticated participants to join this domain and which Reader/Writer has access to what Topic.

DDS built-in AccessControl Plugin, DDS:Access:Permissions [54]. Implements AccessControl by using a permissions document signed by a shared CA. Permission document is an XML format file including Domain rules and Topic rules.

The Cryptographic Plugin defines all cryptographic operations necessary to support encryption, digest, message authentication codes, and key exchange.

DDS built-in Cryptographic Plugin, DDS:Crypto:AES-GCM-GMAC [54]. Provides authenticated encryption using Advanced Encryption Standard (AES) in Galois Counter Mode (AES-GCM). It supports two AES key sizes: 128 bits and 256 bits. It may also provide additional reader-specific message authentication codes (MACs) using a Galois MAC (AES-GMAC).

The Logging plugin is responsible for recording all security-relevant events for DomainParticipants.

DDS built-in Logging Plugin, DDS:Logging:DDS_LogTopic [54]. Implements logging by publishing information to a Built-in Topic named "DDS:Security:LogTopic." All the DomainParticipants need to have access to this Topic, which is predefined in the permissions document.

And finally, the Data Tagging Plugin. Its purpose is to add security tags to data samples. It does not have a DDS built-in Plugin.

Quality of Service

DDS relies on the use of QoS objects to control some aspects of the behavior of the DDS service. A QoS is comprised of individual QoS policies that are represented by objects deriving from QoSPolicy Class.

Any Entity can be associated with a QoSPolicy object such as Topic, DataWriter, DataReader, Publisher, Subscriber, and DomainParticipant. Publishers and subscribers follow a subscriber-requested, publisher-offered pattern. This means that publishers specify an "offered" value for a specific QoSPolicy. The subscriber specifies a "requested" value. The DDS service determines if the two are compatible and if they should initiate communications.

DDS specifies 22 QoS policies [6].

- USER_DATA - Allows applications to attach additional information to the created objects for the use of remote applications discovering their existence. (e.g., security credentials for authentication can be attached).
- TOPIC_DATA - Allows applications to attach additional information to created Topics for the use of remote applications discovering their existence.
- GROUP_DATA - Allows an application to attach additional information to created Publishers or Subscribers for the use of remote applications discovering their existence.
- DURABILITY - This policy controls whether the Service will make data available to late-joining readers.

- DURABILITY_SERVICE - This policy configures the HISTORY QoS and the RESOURCE_LIMIT QOS used by the Entity responsible for persisting information in DURABILITY enabled Entities.
- PRESENTATION - This policy controls the dependencies between data objects that are sent (e.g., order).
- DEADLINE - This policy sets the deadline values for data objects to arrive at subscribers.
- LATENCY_BUDGET - This policy provides a means for the application to indicate the urgency of the data communication.
- OWNERSHIP - This policy controls whether the Service allows multiple DataWriter objects to update the same instance of a data object.
- OWNERSHIP_STRENGTH - This policy is used to determine ownership of a data instance.
- LIVELINESS - This policy controls the mechanism and parameters used by the Service to ensure that particular entities on the networks are still alive.
- TIME_BASED_FILTER - This policy allows a DataReader to indicate that it does not want to see all values published under the Topic. Instead, it wants to see at most once change every separation period.
- PARTITION - This policy allows introducing a logical partition concept inside the "physical" partition induced by a domain.
- RELIABILITY - This policy indicates the level of reliability requested/offered by a DataReader/DataWriter (at-most-once or at-least-once).
- TRANSPORT_PRIORITY - This policy allows applications to take advantage of transports capable of sending messages with different priorities.
- LIFESPAN - This policy defines the time during which the message is valid. Avoiding delivering "stale" data to applications.
- DESTINATION_ORDER - This policy controls how subscribers resolve the final data instance written by different DataWriter objects running on different nodes.
- HISTORY - This policy controls the behavior of the Service when the value of an instance changes (i.e., DataWriter sends a new message) before it is finally communicated to some of its existing DataReader entities.
- RESOURCE_LIMITS - This policy controls the resources that the Service can use to save messages to meet the requirements imposed by the application and other policy settings.
- ENTITY_FACTORY - This policy controls the Entity's behavior as a factory for other entities (e.g., DomainParticipant for Publisher, Subscriber, and Topic).
- WRITER_DATA_LIFECYCLE - This policy controls the behavior of the DataWriter with regards to the lifecycle of the data instances it manages.
- READER_DATA_LIFECYCLE - This policy controls the behavior of the DataReader with regards to the lifecycle of the data instances it manages.

2.2.3 MAVLink

MAVlink is a communication protocol specially designed for Micro Air Vehicles (MAVs). It started being developed in 2009 by Lorenz Meier at the ETH Zürich. Today it has a significant number of contributors and is under the governance of the Dronecode Project.

It supports different transport layers and mediums, like TCP, UDP, or serial telemetry low bandwidth channels operating under the 1Ghz. Currently has two major versions, V1.0 and V2.0. The second version builds upon what the first was, adding features like message signing. 2.0 was the one researched upon.

Architecture

MAVLink network is composed of various systems; they can be vehicles, grounds stations, antenna trackers, and so on. Each with a network-unique id (system id). System ids have a value between 1 and 255. Usually, 1 is for the main drone, but more drones can be added to the network.

Each system is composed of one or more components, in the example of a drone, that can be the autopilot, camera, GPS, or a few others (for more information on drone components, refer to Appendix A - Drones composition). Component id is allocated by type and number from a list called MAV_COMPONENT that MAVLink controls. Some are open for the use of the developer. It has a total of 250 values.

Messages can be intended for all systems, specific systems, all components in a system, or specific components within a system. The protocol defines two 8-bit fields that can (optionally) be specified in the message payload to indicate where the message should be routed. If the ids are omitted or set to zero, the message is considered a broadcast (intended for all systems/components). However, in reality, every message is "broadcasted". MAVLink components are expected to process messages with a matching system and component id and broadcast messages (both set to zero). They are expected to resend messages that are intended for other (or all) recipients to other active channels.

Broadcast messages are forwarded to all channels that have not seen the message. Addressed messages are resent on a new channel if the system has previously seen a message from the target on that channel (messages are not resent if the addressee is unknown or is on the original/incoming channel).

MAVLink systems may be connected across different transports mediums, so Intel created the MAVLink Router [55], which allows to mix-and-match different IP protocols with serial ports and routes MAVLink traffic.

Message Format

MAVLink's message has 10 or 11 sections depending on if it is signed or not. Table 2.12 shows the message format. The first byte is for STX that stands for Start-of-text. This marker is used to indicate the beginning of a new packet. It currently must be set to 0xFD.

LEN indicates the length of the payload section. It must be a value between 0 and 255 due to the field being just 1 byte. INC FLAGS are the incompatibility flags. This indicates how the packet must be interpreted to see if the current application can understand the packet. For example, if the flag is 0x01, it indicates that the packet is signed, and only systems with MAVLink version two will understand it. CMP FLAGS are the compatibility

bit	0	1	2	3	4	5	6	7
Byte 1	STX							
Byte 2	LEN							
Byte 3	INC FLAGS							
Byte 4	CMP FLAGS							
Byte 5	SEQ							
Byte 6	SYS ID							
Byte 7	COMP ID							
Byte 8	MSG ID							
Byte 10								
Byte 11	PAYLOAD (if any)							
Byte 11+N								
Byte 12+N	CHECKSUM							
Byte 13+N								
Byte 14+N	LINK ID							
Byte 15+N	TIMESTAMP							
Byte 21+N								
Byte 22+N	SIGNATURE							
Byte 27+N								

Table 2.12: MAVLink's Message Format (adapted from [18]).

flags. This is used to add information to the packet, so it is not required to be understood and does not prevent the application from processing the message. For example, it can be used to express the priority of a packet.

SEQ is an incremental value that increases with each message sent. It is used to detect packet loss and determine the quality of the connection.

SYS ID is the ID of the system sending the message. COMP ID is the ID of the component sending the message. MSG ID is used to indicate the message type that is in the payload. It informs the applications on how to decode the data in the payload. MAVLink defines over 200 message types.

PAYLOAD contains the message data. The payload section is where the target_id and target_component parameters are. The CHECKSUM is a magic byte to detect message corruption. It has a size of 2 bytes and uses CRC-16/MCRF4XX to calculate the magic byte.

Finally, SIGNATURE is an optional packet used to add security to communications. SIGNATURE is 13 bytes long. 1 byte is the linkID. It is used to indicate the channel in which the packet is sent. The timestamp is 6 bytes and is used to indicate the time past since 1st January 2015 GMT time (to reduce the size required). In the end is the actual signature with 6 bytes, encrypted based on the whole packet, timestamp, and a secret key.

MAVLink defines a set of enums, commands, and message types to use in communications. Enums defines the meaning of specific numbers, commands are drone actions (e.g., fly to waypoint, take-off, and so on), and the messages which might contain commands, some enums, or any other relevant information.

Security

MAVLink's only security option is based on message signing. The secret key must be manually generated and shared via a secure channel to other trusted devices.

Systems must have a shared key outside of MAVLink in order to be able to communicate. A particular case in which the secret key may be shared to other devices using MAVLink is a SETUP_SIGNING message. However, it should only be sent over a secured link like USB or wired Ethernet.

Quality of Service

MAVLink QoS is defined by the type of "Microservice" systems agree with. The MAVLink "Microservices" define higher-level protocols that MAVLink systems can adopt in order to better inter-operate. For example, QGroundControl, ArduPilot, and PX4 autopilots share a common Command Protocol for sending point-to-point messages requiring acknowledgment.

Protocols are used to exchange many types of data, including parameters, missions, trajectories, images, or other files. If the data is too big, it can define how data is split and re-assembled or how to ensure that any lost data is re-transmitted. Other protocols might provide command acknowledgment and error reporting (i.e., expecting either an ACK saying everything is fine or a NAK containing the error). The types of messages used depend on the Protocol agreed.

Using a Protocol that does not expect an acknowledgment, MAVLink offers at-most-once delivery. However, if the system expects a matching acknowledgment, it offers the at-least-once type of delivery because if no ACK is received, it will resend the message.

2.2.4 Honorable Mentions

While analyzing the literature, some protocols that did not quite fit the needs for the project showed up. The protocols are UranusLink, UAVCAN, and MATRIX. These are some open-source protocols intended for drone use that might be useful after they matured for a while longer.

UranusLink

UranusLink [7] is a packet-oriented protocol. Its intended use is in communications between the UAS and the ground control station. It has not been used outside laboratory work.

One of its objectives was to deliver as low overhead as possible. An UranusLink message has six fields. The preamble (PRE). It defines the start of the packet, and the value defined is 0xFD.

The sequence number (SQN) is an incremental value that increases with each message. It is used to detect communication problems so that the Drone might act accordingly. SQN can only hold even values in the range from 0x0000 to 0xFCFE, to avoid the occurrence of 0xFD.

The message identification (MID). It determines how data should be interpreted. There are eight types of messages in the direction of the drones and sixteen in the direction of

the ground station.

PRE	SQN	MID	LEN	DATA ...	CS
1 B	2 B	1 B	1 B	1 – 252 B	1 B

Figure 2.9: UranusLink's Message Format (from [7])

The data length (LEN) carries the length of the DATA field. The data, as such, and has a maximum size of 252 Bytes. In the end, there is the checksum (CS) for integrity.

It can also encrypt its communications using AES, but the DATA payload is truncated to only 29 Bytes.

It is not widely known, as only the original papers and a couple of others researching the protocols, aside from the one only briefly touching the subject.

UAVCAN

UAVCAN [56] is a lightweight open-source protocol designed for reliable intra-vehicular communications. The name stands for Uncomplicated Application-level Vehicular Communication and Networking. It has a specification still in beta [57] at the time of this thesis.

The protocol is divided into two components, the transport layer that can be used over networks such as Ethernet or Controller Area Network (CAN) and a serialization layer based on the Data Structure Description Language (DSDL).

DSDL defines two types of data, which are messages and services. Message types are used to send information over the publish-subscribe style of communications. Service types are used to perform the request-response style of communication, like RPC.

MATRIX

MATRIX [58] is an open-source communication protocol. It aims to offer real-time communication using a decentralized architecture. Although not designed for drones used like the others, drones are one of the examples offered as a possible client.

Its specification [59] is based on JSON over REST to be web-friendly. It defines five APIs, the most noticeably the Client-Server API and the Server-Server API. The APIs send JSON objects, also knows as "events," between the different entities, whether they are clients, servers, or services.

Clients communicate with each other by sending events in the context of a virtual "room." Clients will send an event to their server using the Client-Server API. The server stores the communication history and account information for all of its clients and shares data with other servers that have users participating in the message's room. They share data by synchronizing communication history with other servers (Server-Server API), and those servers send the message to their clients.

Because there is more than one server, no single server is in control over a given room. This process of synchronizing conversation history between servers is called "Federation."

2.2.5 Protocol Comparison

MQTT, DDS, and MAVLink work in very different ways. Table 2.13 shows the literature for this section. The research column shows the literature that helped understand what protocols were used in the context of IoT and learn more about each protocol. Next, the performance tests column shows the literature related to testing the different protocols in terms of communication performance (e.g., their throughput, latency, as so on). Finally, the security column shows the literature related to researching and assessing the security vulnerabilities of the protocols.

Protocol	Research	Performance Tests	Security
MQTT	[51] [52]	[16] [60]	[44]
DDS	[51] [61]	[16] [60]	[62]
MAVLink	[63] [64]	-	[65]

Table 2.13: Sources studying related protocols.

MQTT only supports TCP, DDS supports both TCP and UDP, and MAVLink supports TCP, UDP, and serial connections. Both MQTT and DDS support Publish-Subscribe and asynchronous Request-Response communications. MAVLink supports both publish-subscribe and point-to-point communications. The broadcast messages are considered publish-subscribe because every active system in the network will be interested in the information.

In terms of message overhead, MQTT message size is 1 Byte for the header, plus the variable header. DDS message size is 16 Bytes for the message header, plus 4 bytes for each sub-message. MAVLink has 13 Bytes of header (without security) and a payload that can have a maximum of 255 Bytes.

MQTT has persistence capabilities, enables applications to select what published messages to retain. DDS also has persistence capabilities and allows the configuration of how long the data should be kept by means of QoS policies. MAVLink does not offer any persistence capabilities.

Security-wise, MQTT supports different authentication options, including TLS, SASL, and other challenges/response style authentications. It can use TLS also to encrypt the channel, but there are no mandatory requirements in the specification. [44], as stated in the other comparison, presents five attacks to MQTT.

DDS has DDS:Security, which the bare minimum it offers is authentication for all parties, encryption for the channel, access control for DomainParticipants, and logging. [62] researches on vulnerabilities on DDS, more specifically on the wire-transfer protocol (RTPS). It presents two attacks.

MAVLink only offers message signing as security options. citedominSecurityAnalysisDrone uses fuzzing techniques to discover security vulnerabilities on MAVLink. It manages to find some, in particular, one of them capable of crashing the drone.

MQTT offers 3 QoS options, at-most-once, at-least-once, and exactly once. DDS offers multiple QoS options, one of them being how the message will be delivered (RELIABILITY QoS). However, this option only allows at-most-once and at-least-once. MAVLink also offers at-most-once and at-least-once.

Performance-wise, [16] states that DDS offers low latency no matter the message size; the same cannot be said for MQTT. However, DDS also uses considerably more memory.

In [60], results show that DDS also offers lower latency than MQTT. Unfortunately, no MAVLink performance papers were found.

Finally, MAVLink offers no scalability, only allowing 255 systems. MQTT offers some scalability options, consisting of brokers that decouple communications, and DDS offers excellent scalability due to broker-less architecture and discovery protocols.

Table 2.14 shows a summary of the comparison.

Property	MQTT	DDS	MAVLink
Transport Layer	TCP	TCP,UDP	TCP,UDP,Serial
Message Distribution	Request-Response, Publish-Subscribe	Request-Response, Publish-Subscribe	Point-to-Point, Publish-Subscribe
Message Overhead	1 Byte + Variable Header	16 Bytes + 4 Bytes	13 Bytes
Message Size Limit	No	No	Yes
Message Durability	Yes	Yes, and for how long	No
Security	TLS, SASL, username + password	DDS:Security	Message Signing
QoS Levels	3	22	2
Scalable	With external tools	Yes	No

Table 2.14: Protocols comparison.

2.3 Physical layer communication technologies in drones

In order for drones to communicate with U-Space services, there needs to be a connection to the support infrastructure, which requires a an onboard network device on the drone. Although picking a communication device is outside the scope of this thesis, defining the network profiles of different physical layer technologies is not. These profiles will serve as basis to establish the communication parameter thresholds used during the testing phase to verify the communication robustness under different performances.

The profiles consist of throughput, latency, jitter, and Bit Error Rate (BER). Throughput is the number of packets that the technology can handle per second and will be described in megabits per second. Latency is the delay, in milliseconds, from the message being sent to it being received. Jitter is the variation of latency over time, also described in milliseconds. Finally, BER is the number of bit errors per unit of time and is described in percentage.

The technologies picked are Wi-Fi, Satellite, L-DACS, 4G, and 5G. For the following subsections, various profiles for the technologies picked are shown, which are from the different papers read during the literature review. In the final subsection (Network profiles), the final profiles used for the tests are presented.

2.3.1 Wi-Fi

This communication technology is present in our everyday lives. Wi-Fi stands for wireless fidelity, and it is based on the IEEE 802.11 [66] family of standards that address the WLAN (Wireless LAN).

The standards [67] define the communications by two types of equipment: a wireless station

which is usually a PC/Cell phone equipped with a wireless network interface card, and an Access Point.

Wi-Fi can operate in two modes; one is the infrastructure mode where the access point (AP) is connected to the wired network infrastructure and all the clients connect to the AP.

The other mode is Ad hoc mode (also called peer to peer model) and is simply a set of wireless stations that communicate with one another without using an AP. This means that clients can communicate directly with one another.

Wi-Fi usually uses the frequencies 2.4 GHz and 5 GHz to communicate. Due to the lack of range offered by Wi-Fi, the technology is only helpful in the line of sight (LOS) operations.

Table 2.15 shows different Wi-Fi profile values gathered from different papers. The significant contrast in throughput values can be attributed to the many versions of Wi-Fi that exist.

Paper	Throughput (Mbps)	Latency (ms)	Jitter (ms)	Bit error rate (%)
[68]	2 - 2.20	9 - 17	-	3 - 5
[69]	4.1	30 - 35	4	-
[70]	2 - 10	5 - 15	2 - 3	-
[71]	25 - 200	50	-	-
[72]	54	-	-	-

Table 2.15: Wi-Fi profiles from other papers.

2.3.2 Satellite

Satellite communication is one of the oldest forms of communication we have that is still in use nowadays. It is very useful for long-distance communications because it can operate independently of the distance between the transmitter and the receiver [72].

One of the key oppositions to more use of this technology in drones is the heavy power that the installed module consumes of the drone's energy resources. A resource that UAS hold very special.

Satellite communication is offered in a wide range of frequencies, with frequencies ranging from 5.4MHz to values like 150GHz [73].

Table 2.16 shows different Satellite profile values gathered from different papers. Due to its high energy consumption, it is not really used in the context of IoT or drones; and as such, not much research about it.

Paper	Throughput (Mbps)	Latency (ms)	Jitter (ms)	Bit error rate (%)
[74]	10	275	-	-
[72]	1 - 100	-	-	-

Table 2.16: Satellite profiles from other papers.

2.3.3 L-DACS

LDACS [75] stands for L-band Digital Aeronautical Communications System. It is an air-to-ground communication system thought to fulfill its role as a new standard in the future communications infrastructure for aviation. ICAO started standardization in the end of 2016, and LDACS' draft SARPs (Standards and Recommended Practices) have been developed and endorsed in October 2018.

As the name indicates, it operates within the L-band around 1 GHz (964 – 1156 MHz) range.

Table 2.17 shows different LDACS profile values gathered from different papers. LDACS is a recent technology. Because of that, there is not much research about it.

Paper	Throughput (Mbps)	Latency (ms)	Jitter (ms)	Bit error rate (%)
[76]	0.163	138	-	0.21
[77]	0.300 - 1	-	-	-

Table 2.17: Satellite profiles from other papers.

2.3.4 4G

Presently, this is the most widespread standard used in mobile telephony systems. 4G (LTE) follows the requirements of the IMT-Advanced Standard [78] issued by the International Telecommunication Union (ITU) [79] in 2008.

From the results of a Comparative Study between 3G and 4G [80], we can see that 4G has better overall performance compared to its predecessor, 3G. The theoretical throughput of 3G is only up to 2 Mbps, whereas 4G allows for up to 200 Mpbs.

This allowed for better voice-over IP communications. Although 3G can use VOIP, it was not intended to use it, as it uses both circuit and packet switching techniques. Typically circuit-switched networks are used for voice and packet-switched networks for data. 4G was designed to used VOIP (only packet-switching) and had the performance to allow it.

As mobile communications evolved, the need to allow high-speed vehicles access to the network became a reality. LTE Advanced (LTE-A) is an upgrade to 4G, basically a 4.5G, which requires that the data rates for high mobility objects are at least 100 Mbps, i.e., trains and cars, and 1 Gbps for low mobility objects [81]. It operates in the frequency band from 2GHz to 8GHz [80].

Table 2.18 shows different 4G profile values gathered from different papers. One reason that could explain the profile values disparity can be the use of different 4G technology standards. As time went by, new LTE standards were created.

3GPP release 12 introduced [87] LTE-M or LTE-MTC (Machine Type Communications) and in release 13 [88] Narrow Band IoT (NB-IoT). Both are LTE standards, made explicitly to M2M communications [89].

LTE-M is a low-power broad area network to enable wide-range communications for low-power machines. It offers more data rate and voice over the networks, but it uses more bandwidth. This one was created for outside coverage.

NB-IoT reduces bandwidth, and therefore data rate, in order to reduce energy consumption. Furthermore, And it does not offer VOIP. This one was created for indoor coverage.

Paper	Throughput (Mbps)	Latency (ms)	Jitter (ms)	Bit error rate (%)
[70]	8 - 25	2 - 8	0 - 2	-
[82] Normal	10	14 - 30	3 - 5	-
[82] Edge*	10	48 - 4000	3 - 3.5	-
[83]	10	19	-	-
[84]	15 - 30	25	-	-
[85]	15 - 30	30 - 35	-	0 - 6
[86]*	0.250 - 4	50 - 400	-	0 - 3
[71]	1000	9	-	-
[72]	100 - 300	-	-	-

Table 2.18: 4G profiles from other papers (* Really degraded conditions).

Since there are various LTE standards with different intended scenarios for deployment, there is much research on it. Due to the vast research available, values in both normal conditions and edge conditions are gathered.

2.3.5 5G

5G is an upcoming mobile communication technology that follows the IMT-2020 requirements [90] published by ITU [79].

It was designed to be prepared for something called network slicing [91]. Network slicing is a network architecture that enables the concurrent functioning of different virtual networks in the same physical infrastructure. Each of these slices of the network follows the end-to-end principle, and because of that, can satisfy all the needed requirements for a different application sharing the infrastructure.

3GPP is working on the standardization of 5G, following the requirements of ITU [92]. One of the technologies they are developing, 5G New Radio (NR), provides the capability of using wider frequencies like the millimeter-wave (from 30GHz to 300GHz) [93], in addition to sub 6GHz frequencies. 5G NR is crucial to achieving the IMT-2020 requirements because, as stated in [94], to achieve data rates of multi gigabits per second, communications need to be done in frequencies superior to 24 GHz.

Similar to 4G, it also focuses on allowing fast-moving objects to keep a stabilized internet connection. However, while 4G only supports objects up to 350 km/h, 5G supports communications with high mobility objects with speeds up to 500 km/h [95]. Figure 2.10 shows the main differences between the requirements for 4G and 5G.

Table 2.19 shows different 5G profile values gathered from different papers. Because 5G is still in early deployment, there are not many network performance tests in the context of IoT and drones. That is why the first two rows of the table are from the ITU 5G requirements.

Paper	Throughput (Mbps)	Latency (ms)	Jitter (ms)	Bit error rate (%)
[90] (eMBB)	DL - 100; UP - 50	4	-	-
[90] (URLLC)	-	1	-	-
[96]	1000	1 - 10	-	-

Table 2.19: 5G profiles from other papers.

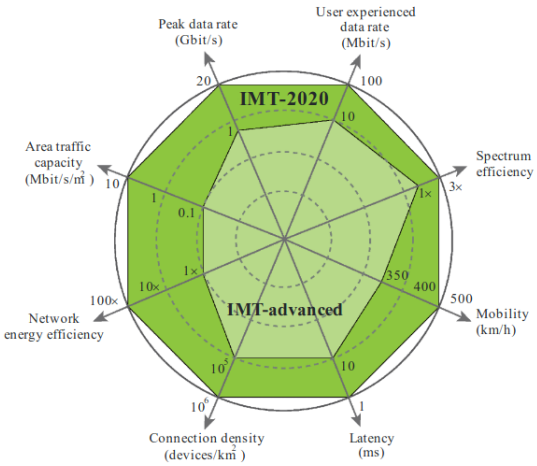

Figure 2.10: capabilities comparison between IMT-Advanced and IMT-2020 (from [8]).

In contrast to 4G, 5G was designed to be deployed in different scenarios that require different performance quality priorities. ITU specifies three main scenarios for 5G capabilities, as shown in Figure 2.11.

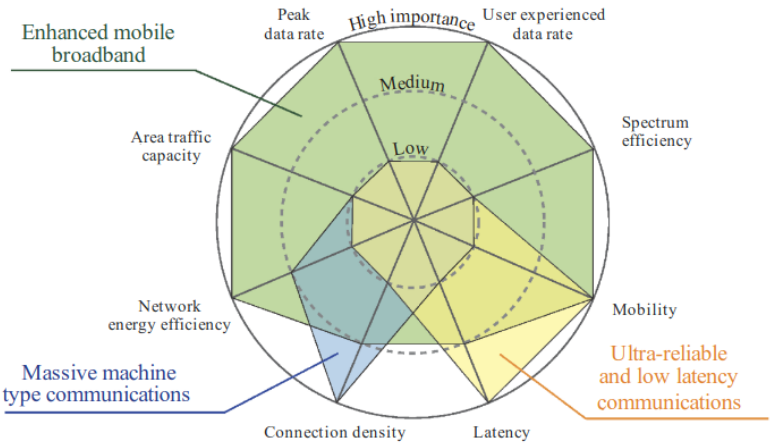

Figure 2.11: 5G scenarios key capabilities (from [8]).

Enhanced Mobile Broadband (eMBB) is focused on user-experienced data rate, traffic capacity, and spectrum efficiency. This is mainly intended for human use and every day-to-day activity.

Ultra Reliable Low Latency Communications (URLLC) is only focused on low latency and low error rate for applications requiring uninterrupted and robust message exchanges (e.g. to enable the safety-critical applications).

Massive Machine Type Communications (mMTC) is focused on high connection density for long periods. This is intended for IoT devices, which generally are deployed in huge numbers in a closed space, producing small amounts for a long period of time.

In the context of drones, depending on their mission, both URLLC and mMTC can be used, but it is clear that URLLC is the way to go for high-risk missions.

2.3.6 Network profiles

Based on the research of the different physical layer technologies, a set of network profiles for evaluation must be selected. The criteria to chose the values were based on the paper's relevance to the thesis context (e.g., [71] and [72] are researches within the context of drones), on how common the values were across papers, and on how different are the profiles from each other. The network profiles are intended to validate the communication system by simulating different network characteristics; having two similar profiles would be the same as having just one.

From the literature survey that was undertaken, there also were not many values for jitter and Bit Error Rate (BER) available. This can be attributed to both performance qualities varying with the status of the communication system at the time of measurement (e.g., the number of clients, deteriorated signals, and so on) However, we can see from the beforementioned values we have that jitter usually is from 0 to 5 milliseconds, and packet loss is from 0% to 5%.

Due to the lack of values for jitter (variance of latency), it was not considered for the current communication platform being developed. It will be tested in a more mature future version of the platform. Packet loss, on the other hand, is very relevant for the context of the project; the values picked were: Packet Loss - [0, 0.5, 1, 5] %

So, in the end, each profile will be defined by its throughput and latency tested with various degrees of packet loss. Table 2.20 shows the values picked for the network profiles. These parameters were chosen in order to spawn an interval that encompasses the values most commonly found in the revised literature.

Name	Throughput (Mbps)	Latency (ms)	Jitter (ms)	Bit error rate (%)
Wi-Fi	50	50	-	-
Satellite	20	200	-	-
L-DACS	1	140	-	-
4G Normal	30	30	-	-
4G Edge	10	100	-	-
5G	100	1	-	-

Table 2.20: Communication platform's network profiles.

These are the network profiles that will be applied during the testing phase (refer to Testing) to understand how the communication system handles different network characteristics.

2.4 Drone simulation

The communication system that is going to be built will be communicating directly with a drone. Testing with a real drone is impractical, thus, a simulation-based approach was chosen.

However, a UAS is a complex machine. The central processing unit, the flight controller, receives information from different sensors and processes the information from within the drone and the environment. Flight controllers refer to the hardware part of the processing unit, and autopilots refer to the software part. In Appendix A - Drones composition it is described the drone constitution and what the autopilot does in further detail. Two

autopilots stand out, ArduPilot and PX4.

These are widely used open-source autopilots that have evolved from simple drone autopilots to many more types of robots. In addition, they both offer simulation capabilities to develop and test things locally. ArduPilot's and PX4's simulation environments will be presented, followed by the comparison of both autopilots using a novel experimental method done in [19].

2.4.1 ArduPilot

ArduPilot is an open-source autopilot software system. It was created in 2009 as an autopilot for drones and has matured, offering piloting capabilities to more vehicle systems, including sailboats, powered boats, submarines, and Balance-Bots. Its homepage [97] states that it is installed in over 1 million vehicles worldwide. It also states that known organizations like Nasa, Intel, and Boeing use ArduPilot for testing and development purposes, as well as universities around the world.

In terms of simulation, it offers both SITL and Hardware in the Loop (HITL). When simulations are executed in SITL, everything is software-based. The drone, the environment, and the sensor data. All being executed on the user's computer. When simulations are executed in HITL, the environment and sensors are still software-based, but all drone instructions are executed on a real drone's hardware, connected to the user's computer. Note that it still is a simulation, and the drone does not actually fly. Nevertheless, it has the benefit of testing it in real hardware.

Figure 2.12 shows ArduPilot's SITL architecture.

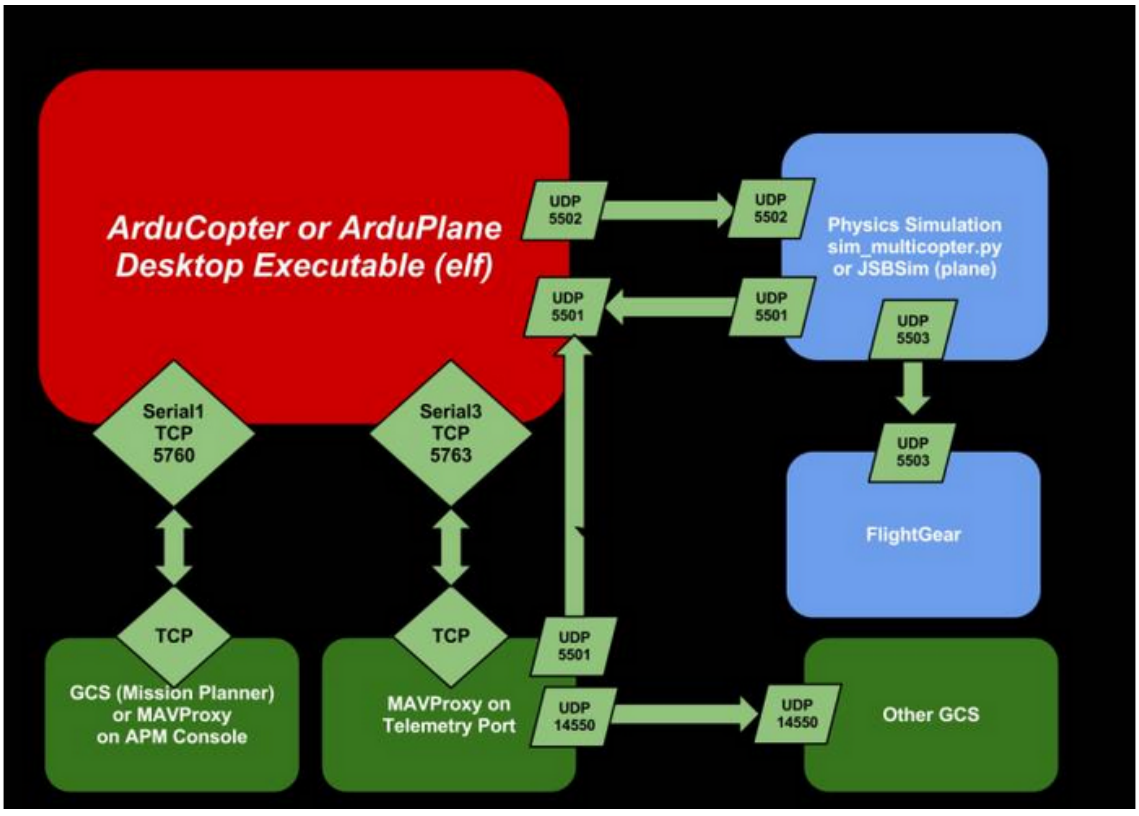

Figure 2.12: ArduPilot's SITL architecture (from [9]).

ArduCopter is the autopilot. It acts as the brain of the drone. For it to work, it needs to receive information about the physics involved with vehicle movement.

A Flight Dynamics Model (FDM) of the vehicle is necessary to produce this information. Simulators can offer FDMs for different vehicles. ArduPilot offers compatibility with any simulator that uses MAVLink to communicate, like the sim_multicopter and JSBSim. The autopilot receives sensor information from the simulator like yaw, roll, pitch, velocity, and many others, so it knows what to do to keep the drone stabilized. The autopilot sends actions for the drone's model in the simulator, with the necessary actions to conclude a mission or stabilize the drone (e.g., motor control actions).

GCS is the Ground Control Station. It is used to send commands to the drones. Can send commands such as take-off, fly to waypoint, and mission plans (a group of ordered waypoints). ArduCopter ground station of choice is Mission Planner. It communicates via MAVLink, so any other Ground Station that uses MAVLink is compatible. One of them is MAVProxy, a minimalist ground station capable of forwarding messages over UDP networks to other ground stations.

FlightGear is an optional add-on that allows the user to see the drone in a 3D environment vs. the 2D image provided by the Mission Planner.

2.4.2 PX4

PX4 is also an open-source autopilot software system. It was also created in 2009, the result of a Micro Air Vehicle competition in Europe. As ArduPilot, it matured over the years and now supports more vehicles like rovers, boats, and submersibles.

PX4 belongs to Dronecode Foundation, which aims to set the standards with open-source in the drone industry. Many organizations are members of this foundation [98], being most noticeably Microsoft.

In terms of drone simulations, PX4 also supports SITL and HITL. In addition, there is a specific simulation type for quadrotors called Simulation-In-Hardware (SIH). When simulations are executed in SIH, everything is running on the hardware. The computer is just used to display the virtual vehicle.

Figure 2.20 shows PX4's SITL architecture.

PX4 SITL mode works similarly to ArduPilot's. It has the PX4 (autopilot) communicating with a simulator via MAVLink; following the same principle, PX4 receives sensor data from the simulator and sends commands for the virtual drone. PX4 recommends using Gazebo as the physics simulator.

The ground stations also communicate via MAVLink with PX4, and the recommended software is QGroundControl. It also is prepared to work with offboard control from a companion computer, usually connected via serial port or Wi-Fi, also communicating using MAVLink.

2.4.3 Comparison

There is not a lot of research and tests into drone simulation and autopilots. A recent article [19] tested ArduPilot against PX4 in a Qualitative and Quantitative analysis within the context of UAS.

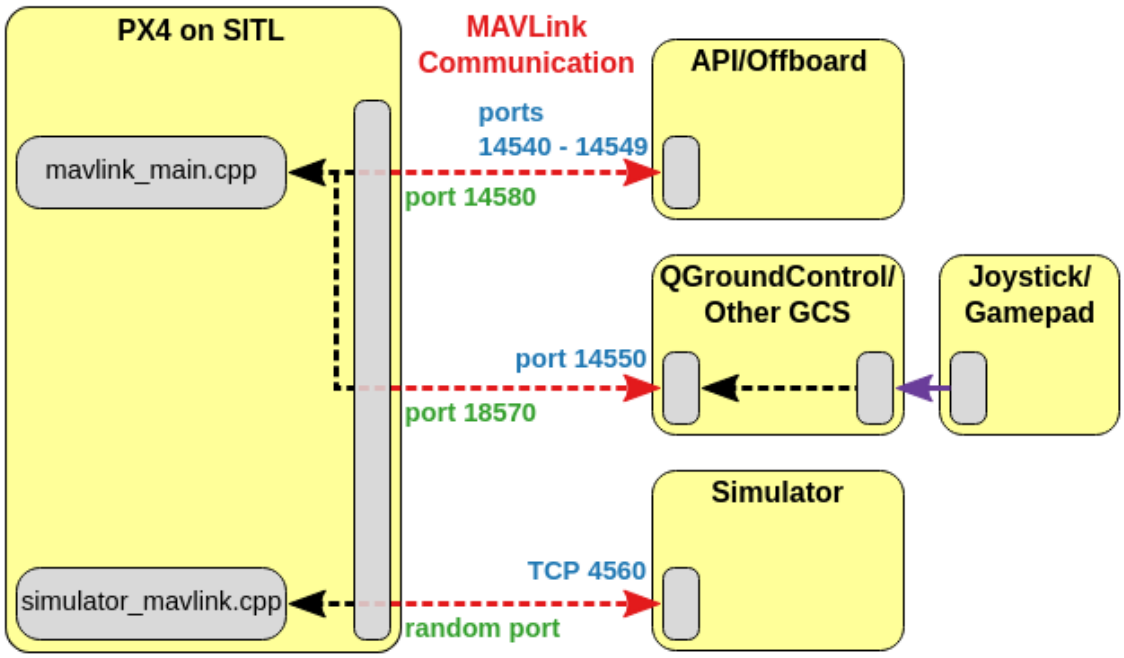

Figure 2.13: PX4's SITL architecture (from [10]).

The Qualitative analysis includes hardware compatibility, user experience, licensing, and partner companies. Hardware compatibility refers to the numbers of different UASs compatible with both ArduPilot and PX4. User experience accounts for different interface aspects, including the calibration process, parameters guideline, and user interface (UI). Licensing refers to how the software can be modified and redistributed. Finally, partner companies account for the companies working together to add value to the autopilot's existence and improvements in the future.

The Quantitative analysis focuses on mathematical and statistical methods to evaluate data points. The analysis includes GPS and Barometer Calibrations errors, Flightpath in terms of time and accuracy, the drone's stability in Loiter mode (hovering), Return to Launch (RTL) flight duration and deviation from the path, Altitude elevation accuracy, and Battery Management of the software.

Table 2.21 and 2.22 show the summary of the comparison for Qualitative and Quantitative analysis.

Qualitative Test	ArduPilot	PX4
Hardware Compatibility	X	
Licensing		X
Partner Companies		X
User Experience	X	

Table 2.21: [19] qualitative test results.

In terms of hardware compatibility, ArduPilot offers compatibility with 21 open and closed drone hardware, while PX4 only offers compatibility with 16. ArduPilot's license focuses on "user experience and distribution of businesses and closed source hardware." In contrast, PX4's license focuses on the "proper distribution of the software."

PX4 has fewer partner companies, but in [19] they conclude that the PX4 partners weigh

Quantitative Test	ArduPilot	PX4
GPS & Barometer Accuracy		X
Flight Path Accuracy		X
Flight Path (Time)		X
Altitude Control		X
Loiter	X	
Return to Launch		X
Battery Management		X

Table 2.22: [19] quantitative test results.

more than the ArduPilot's ones. User experience, although PX4 has a better calibration process, ArduPilots takes the win because it offers a better UI and needs minor Parameter Tuning, and offers more flight modes.

For Quantitative tests, both GPS and barometer module tests showed that the PX4 error rate was lower on both (0.24% 2.6% vs. 0.87% 6.7%). Flightpath accuracy tests showed PX4's slightly better accuracy (0.55% vs. 0.7%), and it executed the mission, on average, 0.476 seconds faster.

Altitude control tests show that PX4 has a better performance compared to ArduPilot. PX4 can elevate to different heights accurately, while ArduPilot elevates to heights with more significant discrepancy. However, ArduPilot maintains its elevations much better than PX4, the latter spiking variously while hovering.

The RTL flight tests conclude that PX4 wins, with the flight being more precise and faster. ArduPilot slows down while descending in altitude in the 10-meter mark, while PX4 maintains speed throughout descending.

Finally, the Battery Management tests reveal that ArduPilot consumes more battery in-flight, allowing PX4 to have longer flight times.

2.5 Related projects & Methodologies

In the U-Space context, various projects and methodologies are relevant for this thesis's work.

CORUS, a SESAR H2020 project, which works resulted in the CONOPS, the concept of operations for drones, and MEDUSA methodology, which builds on top of SORA to deliver a safety assessment of U-Space. ConOps is considered a living document, which means it is continuously updated. Therefore, BUBBLES will use work done by CORUS, evolving the ConOps to elaborate more detailed operations, and MEDUSA as the methodology to validate the BUBBLES solution.

Another Project is also relevant, DroC2Om; also an H2020 project which was tasked with developing a Command and Control communication system for drones. Although not used in BUBBLES, its works were used as research for the communication system.

The next subsections will provide more detailed insights about related projects which may somehow intersect with the scope of the BUBBLES domain.

2.5.1 CORUS

Concept of Operation for European UTM Sytems (CORUS) is a H2020 project within the context of U-Space. Its primary focus is to deliver detailed definitions of VLL UAS operations and the services necessary for operations to be safe and efficient.

The work from the CORUS Consortium led to the creation of Concept of Operations (ConOps) for U-Space. ConOps define clear use cases for nominal scenarios and describe how safety losses in non-nominal situations (i.e., contingency or emergency) can be minimized. Its operations address drone operations in uncontrolled airspace and in and around controlled or protected airspace (e.g., airfields).

To understand the risk posed by the drone missions in different categories of operations, they used Specific Operation Risk Assessment (SORA). However, it does not consider the risk from drone traffic in the context of U-Space, so CORUS proposed a Methodology for the U-Space Safety Assessment (MEDUSA), which uses SORA as a building block.

In addition, they also considered non-aviation aspects, identifying critical issues for society (e.g., safety and privacy, noise) and suggesting solutions to increase social acceptance.

2.5.2 CONOPS

ConOps follow risk and performance-based approach [99]. The Risk approach part means that the level of effort devoted to maintaining safety is proportional to the risk of not doing so. This approach goes in line with EASA regulations. The performance-based approach part means that airspaces have a minimum required performance criteria for UAS to fly in them.

ConOps divide the VLL airspace into three different airspace types - X, Y, and Z. The three airspace types are attributed based on the expected number of drones, ground risk, air risk, nuisance, security or other public acceptance factors, and the U-space services needed to enable safe operations. X expecting less risk and lower demand for U-space services, and Z the highest risk, with high-density operations and need more U-Space services (e.g., separation management). Describing the different U-Space services that need to be provided in them, what types of operations can be performed, and the different requirements for accessing each type.

The types of operations are based on the EASA categories for drone operations, which are Open, Specific, and Certified [99]. Open categories are for relatively untrained and inexperienced pilots, UAS incapable of submitting position reports, and operations like "follow-me" mode (e.g., the drone follows the GPS of a mobile phone). This category is expected to be commonly found in X volumes.

Specific categories are for operations, including the following conditions: it follows a standard EASA scenario, if a SORA has been made for the operation, or if the operator holds a UAS certificate.

Certified operations are for operations of the risk equal to manned aviation. The UAS will always need to be certified, will need air control approval issued by competent authorities, and the operator is required to hold a pilot license.

ConOps also defines that the categories requiring separation assistance are specific and certified, and CORUS assumes that they are indistinguishable in terms of traffic management.

2.5.3 SORA

SORA is a methodology developed by Joint Authorities for Rulemaking on Unmanned Systems (JARUS) [100]. JARUS is a group that contains 63 countries, including Portugal, and two organizations, EASA and Eurocontrol.

SORA is a methodology for risk assessment to supports authorities A methodology for risk assessment in UAS operations within the Specific category. It outputs a Specific Assurance and Integrity Level (SAIL) that determines the necessary Operational Safety Objectives (OSO) to execute the operation with an acceptable level of risk.

The combination of Ground Risk Class (GRC) and Air Risk Class (ARC) values leads to the final rating of the mission, so-called SAIL.

The ground risk is related to the risk of a person, property, or critical infrastructure being struck by a UAS and therefore considers the operating environment with respect to the population density, the type of operation, and the UAS size.

The determination of the air risk considers the probability of encountering manned aircraft in the airspace, which is mainly derived from the density and composition of manned air traffic in the airspace.

Since the SORA approach focuses on single-mission risk assessment, CORUS proposed MEDUSA to conduct the safety assessment methodology of U-Space, considering the outcome of different SORA assessments.

2.5.4 MEDUSA

The MEDUSA [101] is a method proposed by the CORUS team for identifying and managing hazards posed by drone traffic in the U-Space. MEDUSA takes a holistic approach to the U-Space safety assessment. It sees the system as the combination of equipment, procedures, and human resources that together function in the context of U-Space. In contrast to SORA, which only considers the operators' viewpoint.

U-Space cannot simply be considered intrinsically safe when no failure occurs. Because of that, a safety assessment for U-Space will require a success-based approach [91], where everything goes perfectly, to understand if the U-Space services are still capable of offering a tolerable level of safety.

In addition, a failure-based approach is also necessary to test the success of the solution when errors or failures occur. It is concerned with achieving the required level of safety with problems induced by the U-Space services or operations failing.

MEDUSA's process to evaluate the U-Space operations safety consists of 4 steps. The Definition phase (DP), the Operational Specification phase (OSP), Safety assessment at design level (SADL), and Safety assessment at VLD level (SAVLD).

The first phase is to identify the operational environment where drone operations will be conducted, UAS operations and their categories (SORA), risks inherent to those operations, U-space services, and conduct an initial assessment on the U-space services to mitigate risks and identify missing services. DP ends with the definition of relevant U-Space Safety Criteria.

The OSP starts by conducting an early operation level safety assessment to show that the U-Space services mitigate the risks inherent to UAS in normal conditions (success

approach) and satisfy the Safety Criteria from DP. Followed by the identification of relevant abnormal conditions for the environment and UAS, and operational hazards caused by failures of the U-Space services, and assess the severity of the effects. In the end, the safety assessment at the operational level must be done considering normal, abnormal, and faulted conditions, and all satisfy the Safety Criteria.

SADL sees if design requirements meet safety specifications for normal, abnormal, and faulted conditions. At this stage, it is shown that the proposed design supporting U-Space Services is able to mitigate risks in all conditions and satisfy the Safety Criteria.

The VLD in SAVLD stands for Very Large-scale Demonstration. This last phase verifies that the U-Space services assessed during VLD have been implemented according to safety design requirements derived from SAVL. Also, define start/stop criteria to revert back to "normal" operation in case of an unsafe situation during the demonstration. And finally, verify that the U-Space services at this level satisfy the Safety Criteria.

MEDUSA's ultimate objective is to define a complete set of Safety Requirements for the U-Space service implementation and associate these requirements to mitigation actions that will maintain the level of safety equal to for manned aviation in both air and ground.

2.5.5 DroC2om

DroC2om is an H2020 U-Space project from 2017. It has already finished, with the final public deliverable being published on 4th July 2019.

The project focus was on Command and Control (C2) capability for drones. It designed and evaluated an integrated cellular-satellite system architecture concept for data links (C2) to support reliable and safe UAS operations.

It defined a set of scenarios and requirements for combined cellular and satellite datalink communications architecture for C2. Requirements for the datalink ensure that C2 datalink information can be reliably transferred between the U-Space controller and the rest of the UAS system.

It also defined a software-based evaluation environment (system level) for experimental radio investigations. The DroC2om project researched the combined cellular satellite radio network architecture and radio mechanisms necessary to ensure the requirements were successively achieved. This work contributed to some of the 3GPP work involving UAS and is compatible with LTE-Advance and 5GNR technologies (both by 3GPP).

Although the DroC2om communication (C2) and the one required for BUBBLes (distribution) are significantly different, it is important to research the architecture and requirements concluded for the project [20] and understand what is expected of a SESAR project and communication performance related to UAS operations.

Although it does not match the communication service needed for BUBBLES, it is interesting to see architecture and requirements in here . What is expected of a SESAR project and communication times related to UAS.

This page is intentionally left blank.

Chapter 3

Project planning and methodologies

This chapter is dedicated to present the work plan, but also to provide insights about the project development and strategy. It will cover the methodology selected for the solution development, the tasks that composed the work, and a comparison between the planned work and the work done at the end of the project.

The time stipulated for the thesis was ten months, divided into two semesters. Commonly the first semester is focused on the research work and the second semester on the implementation of work laid after research. With not much option to deviate from this time window, there needs to be a clear definition of the steps and dependencies needed to finish the project.

3.1 Methodology

Selecting a good software development life cycle management strategy is extremely important for the development of any piece of software. It is the framework that lays the group of steps, which inform how to develop and maintain the software during its life cycle. The objective of any methodology is to simplify the process of development and concluding projects.

Another thing to note about the project is that once its requirements are revealed, they will not change midway through the process of developing the solution. So, when they are revealed, they will be set in stone.

The specific time frame given to developing the solution and the lack of changes to the requirements makes the Waterfall methodology a perfect fit for the project's needs. This approach is based on a sequential interlink between the software lifecycle stages, in such a way that each phase cannot be started until the prior one is concluded but, at the same time, there is the possibility of refining prior stages if necessary.

There will be one exception to this linearity in the flow of both the implementation and testing phases. The testing phase will follow the development phase, and instead of continuing the sequence, it will loop back to development.

This flow in the opposite direction will allow for the revision and correction of problems found through testing. The fact that multiple test phases will follow the development phase will help find bugs and errors. Finding these bugs and errors early on decreases the amount of effort and time to fix them, and multiple test phases reduce the possibility of

the error propagating through time.

From a development point of view, the git tool was used for version control and support, enabling the establishment of a development cycle akin to a CI/CD pipeline, automating deployments and code builds which are supported by a centralized code repository. All the necessary elements (such as service configurations or code) are obtained in their most up-to-date version from the centralized repository. Moreover, this approach also favors a RERO (Release Early, Release Often) approach while also safeguarding access to the historical log of development deltas.

3.2 Tasks

Table 3.1 shows the state after the intermediary report. Some tasks were completed by the end of the first semester, and the others were planned to be done during the second semester. The first tasks done by the end of the first semester were focused on the research and survey on the literature available. With the research done, the work planned was to read INDRA's documentation, which is working alongside UC in BUBBLES.

Name	Status
Research the project	Completed
Research drone basics	Completed
Research communications network	Completed
Research communication protocols	Completed
Research message brokers	Completed
Research advanced topics	Completed
Read Indra's documentation	To Be Done
Gather requirements	To Be Done
Create architecture	To Be Done
Develop the service	To Be Done
Create test environment	To Be Done
Define test scenarios	To Be Done
Run tests	To Be Done
Write thesis	In Progress

Table 3.1: Project-related planned tasks (status at the end of the first semester).

After having a good understanding of the current state of things, the requirements were planned to be gathered, followed by the design of the solution architecture. With the architecture, implementation, and testing follow. Testing and development of the service go hand in hand so that bugs and errors in the code are found sooner. The test environment creation and testing scenarios (the ones that are tasks) are different from the testing during the development phase, as test tasks are to demonstrate the service's capabilities and do not have the objective of finding any errors.

Throughout the thesis timeline, its writing will be done simultaneously with the tasks of the project, as the chapters are related to the various steps of the project.

Table 3.2 shows the complete set of tasks by the end of the project. In addition to the planned tasks, two were added. Develop MQTT client and Re-write the state-of-the-art.

The MQTT client was needed to integrate with INDRA's software.

Name	Status
Research the project	Completed
Research drone basics	Completed
Research communications network	Completed
Research communication protocols	Completed
Research message brokers	Completed
Research advanced topics	Completed
Read Indra's documentation	Completed
Gather requirements	Completed
Create architecture	Completed
Develop the service	Completed
Develop MQTT client	Completed
Create test environment	Completed
Define test scenarios	Completed
Run tests	Completed
Write thesis	Completed
Re-write State-of-the-art	Completed

Table 3.2: Project-related actual tasks.

Re-writing the state-of-the-art was needed because the first one had no basis for the rest of the work.

3.3 Comparison

Here are presented the tasks mentioned in section above, organized into the thesis' time frame, with the dependencies between each other and time windows for each. Figure 3.1 shows the Gantt of the work done until the end of the first semester and the planned work of the second semester in light blue. Figure 3.2 shows the Gantt of the actual work done by the end of the project.

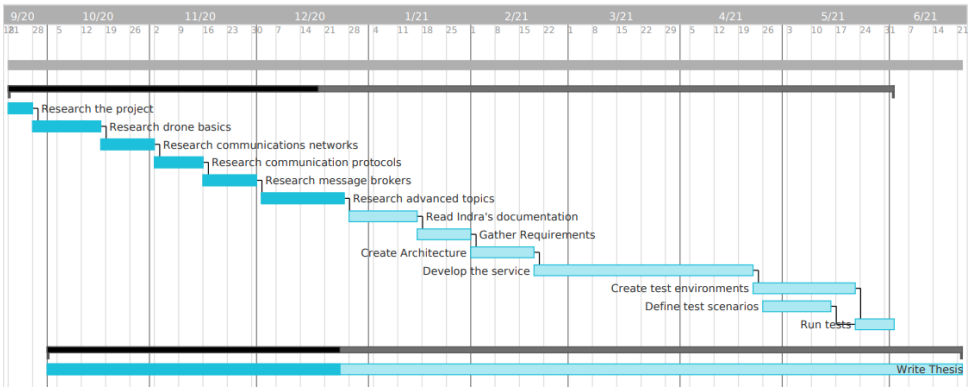

Figure 3.1: Planned Gantt at time of the intermediary Report.

There are three main differences between the two Gantts: the time that took the development service task, the time it took the test environment task, and when the writing of the thesis ended.

The deviations started with Indra's documentation, which was thought to clarify the

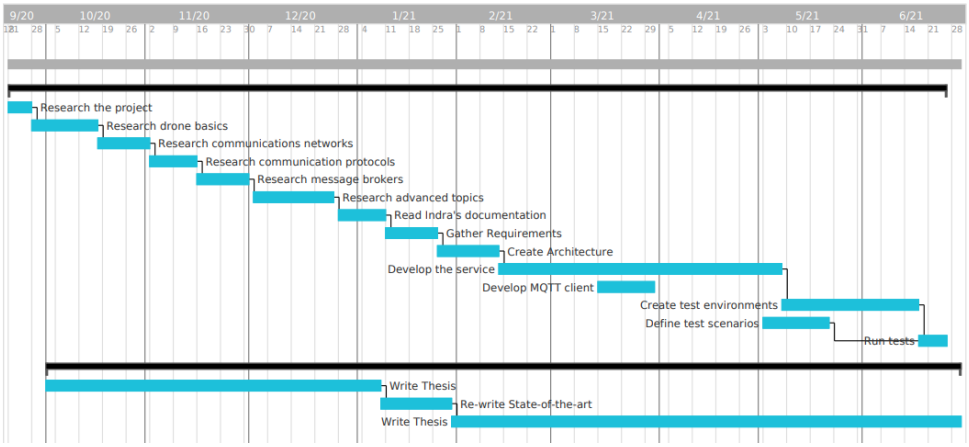

Figure 3.2: Project's actual Gantt.

project's requirements, actually was a set of specifications for drone telemetry and MQTT connection details. Even with this surprise, the requirements gathering and architecture design stages took approximately the same time to conclude. The research on past projects and BUBBLES revealed what the requirements would look like.

During the service development, it was the most noticeable difference that the MQTT client provoked. The MQTT details were to connect to one of the U-Space services, a drone tracker. So an MQTT client had to be developed.

Another note was the definition of the test scenarios task overlapped the end of the development task. This was because the test scenarios picked could impact the configuration of the project. The test scenarios did not impact the development of the service. Still, they influenced the creation of the test environment as it required additional components to run the tests, which added approximately one week and a half to the time needed to finish the project.

Finally, the re-writing of the state-of-the-art started at the beginning of the second semester. Since this chapter is the basis for the rest of the thesis, it had to be changed right away. This moved the end of the thesis to almost the end of June, close to the delivery limit.

3.4 Threshold of success

A Threshold of Success (ToS) is a technique used to define a clear picture of what success is considered for the project. Projects define a minimum set of conditions that, if they are not met, the project is considered a failure. So that in the end, it can be said that this project was a success, a Threshold of Success (ToS) is needed.

The set of conditions defined were:

- According to the MoSCoW ranking methodology, the requirements classified with Must Have must be successfully implemented (refer to Section 4).

- The defined objectives must be completed within the planned deadline.

The next chapter will delve into the details of the requirements elicitation and enumeration process, which are instrumental to the establishment of the aforementioned ToS criteria.

This page is intentionally left blank.

Chapter 4

Requirements

This chapter is dedicated to a detailed explanation of the project's most relevant characteristics and requirements. It will start with a brief description of the project involving the thesis (BUBBLES) and its characteristics and needs. Following will be the requirements for the communication system, both functional and non-functional requirements.

4.1 BUBBLES Characteristics

BUBBLES aims for a future where the sky is filled with drones. To achieve this objective, various systems doing different things need to interact with each other. BUBBLES concept architecture is shown in Figure 4.1. For more on how the service work, refer to Appendix B - BUBBLES services.

For BUBBLES services to work, the drones need to be communicating with these services. For that to happen, there needs to be an infrastructure capable of sending messages between drones and BUBBLES services. Both the drone and its communications capabilities are use to determine the type of bubble the drone will need.

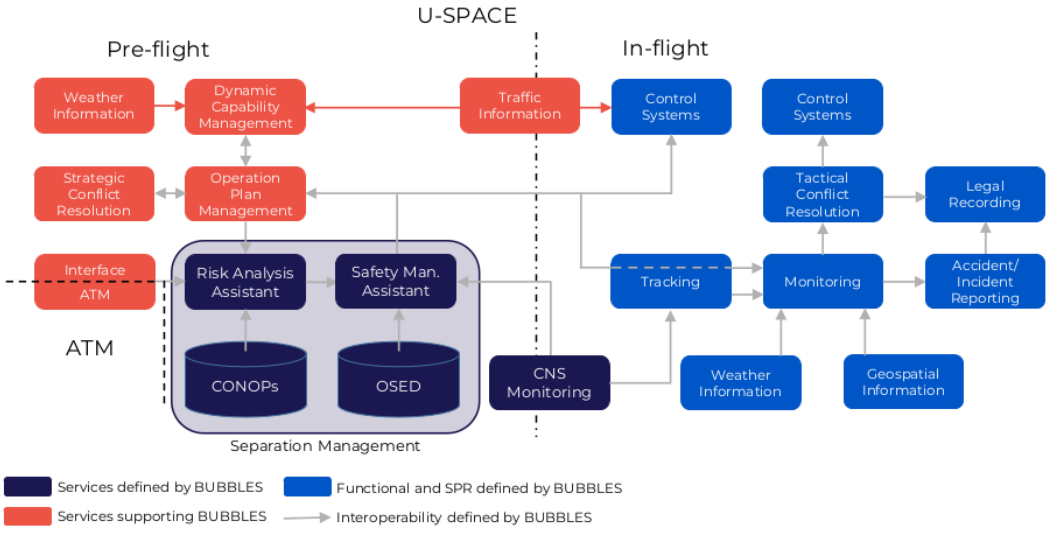

Figure 4.1: BUBBLES's Concept Architecture.

BUBBLES is based on the concept of Performance-Based Navigation (PBN) and Performance-

Based Communications and Surveillance (PBCS).

PBN describes requirements navigation performance (RNP) specification regarding accuracy, integrity, and continuity for the aircraft's positioning and measurements.

PBCS concept provides objective operational criteria to evaluate different communication and surveillance technologies. By defining the Required Communication Performance (RCP) and Required Surveillance Performance (RSP) specifications.

All these specifications define the performance metrics, as well as acceptance thresholds that the air vehicle must comply with to fly in designated airspaces.

Both concepts are used in manned aviation in ATC. Civil Aviation Organization (ICAO) has defined RNP, RCP, and RSP for manned aviation. Neither the concept of PBN nor that of PBCS has yet been applied to UAS. BUBBLES intends to extend the concepts to Unmanned Aerial Systems (UAS) and develop new RCP, RNP, and RSP specifications for UAS.

The objective is to base the specifications on the Methodology for the U-Space Safety Assessment (MEDUSA) (Sub-section 2.5.4). The safety assessment will include the provision by the U-Space of specific separation minima and methods and will lead to a set of CNS required performance specifications.

The communication system should also not limit the number of systems talking to each other; in other words, the system should be scalable.

In terms of security, the communication channels should all be encrypted. In addition, only authorized clients should be able to communicate, with only a select few being able to read or write messages.

Finally, there still some doubt about what architecture to use on U-Space services. Although Air Navigation Services is a centralized market in manned aviation, the U-Space Services provision is not explicit yet. Two scenarios are possible: a centralized model (similar to manned aviation) or a federated one (where different providers seamlessly provide the same U-Space service to different users).

4.2 Requirements

Specific requirements for the system must be identified, both functional and non-functional. This section presents the requirements, which are ranked using a technique called "MoSCoW" [102]. This decision was backed by the need for a simple yet efficient way to prioritize and organize requirements.

Must Have - These requirements provide the Minimum Usable SubseT (MUST) of requirements which the project assures it will deliver. It can be defined as the set of needs that there is no point in delivering the solution without them.

Should Have - These requirements can be defined as important but not vital. It may be hard to leave out these from the solution, but it still would be viable.

Could Have - These requirements can be defined as wanted or desirable but not as crucial as should have requirements, and less impact if left out.

Won't Have - These are the requirements that will not be delivered in the solution.

4.2.1 Functional Requirements

Due to the compliance with BUBBLES' needs, the functional requirements are pretty straightforward. The must have requirements are essential to the functioning of BUBBLES services, i.e., service should be able to interact with each other, as well as with drones, and provide communications for the whole flight, as well as before take-off and after landing.

The should have requirements are the BUBBLES' needs that without them, the solution would still be viable, i.e., the system compatibility with both centralized and federated architecture, multicast communications, and both authentication and encryption.

The could have requirements are the ones desirable but not as crucial as the should have, i.e., limit access control for authenticated users and support point-to-point communications.

Finally, the won't have requirements are the ones outside the scope of this thesis, i.e., the relay of messages of ATC data and voice for part of its serviced users.

The functional requirements of the system are as follows:

ID	Description
RF1	The communication service **MUST** allow BUBBLES's services to interact with each other.
RF2	The communication service **MUST** allow UAS to interact with the BUBBLES services.
RF3	The communication service **MUST** provide communication links for the whole duration of flights as well as prior to take-off and after landing.
RF4	The communication service **SHOULD** be compatible with both ways in which the BUBBLES' services may be organized (centralized or federated).
RF5	The communication service **SHOULD** support point-to-multipoint (multicast) data communications.
RF6	The communication service **SHOULD** encrypt all communications.
RF7	The communication service **SHOULD** authenticate users.
RF8	The communication service **COULD** limit the access permission for authenticated users.
RF9	The communication service **COULD** support point-to-point (unicast) data communications.
RF10	The communication service **WON'T** support the relay of ATC data and voice services for part of its serviced users.

Table 4.1: Functional requirements of the project.

4.2.2 Non-functional Requirements

BUBBLES plans to develop the CNS requirements; but the CNS, and more specifically, the RCP specification will not happen in the thesis time window. Because of this, the non-functional requirements will be based on the ICAO's RCP specifications and DroC2om (see subsection 2.5.5) performance requirements.

Table 4.2 shows the performance requirements for DroC2om.

ICAO for each RCP type sets four different values [21]. They are transaction time, conti-

Req Reference	Requirement description
WP2-GENUS-PER-001	The System shall offer, for all addressed data exchanges, an end-to-end availability of provision of at least 99.3 %.
WP2-GENUS-PER-002	The System shall offer, for all addressed data exchanges, an availability of use of at least 99 %.
WP2-GENUS-PER-003	The System shall offer integrity performance in terms of packet error. rate measured at the interface between network and logical link layer of at least 10^{-3}.
WP2-GENUS-PER-005	The System shall not limit the number of drones supported and/or air-ground data throughput compared to the services offered by the underlying supported data links (with the exception of the throughput limitation resulting from tunneling overhead if used).
WP2-GENUS-PER-006	The System shall not limit the capacity to accommodate a growth of traffic offered by the underlying data links.

Table 4.2: DroC2om requirements (from [20]).

nuity, availability, and integrity.

Transaction time - The maximum time for completing the operational communication transaction, after which the initiator should revert to an alternative procedure.

Continuity - The probability that an operational communication transaction can be completed within the communication transaction time.

Availability - The probability that an operational communication transaction can be initiated when needed.

Integrity - The probability that communication transactions are completed within the communication transaction time with undetected error.

Table 4.3 shows the RCP ICAO has defined for manned aviation.

RCP Type	Transaction time (ms)	Continuity (%)	Availability (%)	Integrity (%)
RCP 10	10	0.995	0.99998	10^{-5}
RCP 60	60	0.99	0.9995	10^{-5}
RCP 120	120	0.99	0.9995	10^{-5}
RCP 240	240	0.99	0.9995	10^{-5}
RCP 400	400	0.99	0.999	10^{-5}

Table 4.3: ICAO'sRPC for manned aviation (from [21]).

DroC2om performance values for availability and integrity are both less strict than the lowest value for ICAO RCP, which is RCP 400 with the availability of 99,9% and integrity 10^{-5}. The ConOps for U-Space also states in [103] that the validations in required navigation performances will follow the same principles as ICAO.

For those reasons, the non-functional requirements rankings are as follows. The must have requirements are scalability, not limiting the number of drones or throughput of data, and RCP 400. Without these three requirements, the solution would not be viable within the context of BUBBLES.

As the solution should be at least TRL 3 solution, the communication system should be

able to follow the RCP 240 and 120 specifications.

The Could have requirements are the last two RCP, and the system won't offer any non-repudiation. The system will be incapable of confirming that a specific user or machine was the actual sender of the message.

ID	Description
RNF1	The System **MUST** not limit the number of drones supported compared to the services offered by the underlying supported data links.
RNF2	The System **MUST** not limit air-ground data throughput compared to the services offered by the underlying supported data links.
RNF3	The System **MUST** offer a communication performance conformance with the RCP 400.
RNF4	The System **SHOULD** offer a communication performance conformance with the RCP 240.
RNF5	The System **SHOULD** offer a communication performance conformance with the RCP 120.
RNF6	The System **COULD** offer a communication performance conformance with the RCP 60.
RNF7	The System **COULD** offer a communication performance conformance with the RCP 10.
RFN8	The System **WON'T** offer non-repudiation for communication publishers.

Table 4.4: Non-functional requirements of the project.

4.3 Summary

This chapter detailed the requirements for the communications platform, defining ten function requirements and eight non-functional requirements. The next chapter will delve into the details of the architecture design and creation process, which will use the requirements to establish the basis for the communication platform architecture.

This page is intentionally left blank.

Chapter 5

Architecture

This chapter is dedicated to the description of the project's architecture. The architecture to be introduced results from the research done in the first semester and the requirements gathered stating the project's needs.

The following sections present the architectural drivers, the communication system architecture built from said drivers, and the description of each architecture component.

5.1 Architectural Drivers

Architectural drivers are requirements that shape the architecture. Therefore, drivers should have a profound impact on the architecture and a high value for the project.

A utility tree registers all the drivers in one place. It defines the priority of each driver based on the impact of the architecture (High, Medium, Low) and the value for the project (High, Medium, Low).

Based on the requirements outlined in Section 4, the utility tree (see Table 5.1) was created.

The utility tree shows that availability, scalability, and compatibility are the qualities that weigh the most in terms of relevance to the project and impact on the architecture. Allowing an unlimited number of drones and maximizing the time that U-Space services are available are essential values for the U-Space vision and require an infrastructure capable of handling multiple connections at the same time while managing problems and hazards without disturbing performance.

In addition, the need to cover both centralized and federated U-Space services changes the architecture drastically and accommodating it is essential to deliver the BUBBLES communication system. Because of this, the architecture can be designed around these drivers and deliver the requirements for the BUBBLES project.

5.2 Communication Services Architecture

With the information from the Architectural Drivers (Section 5.1), a system can be designed. A simplified concept of the architecture follows, which will be built upon in the following subsections

-	Quality Attribute	Attribute Refinement	Driver
Utility	Performance	Transaction Time	All communications should reach its destination within a specific time limit. (H, M)
	Performance	Scalability	The communication system should not limit the number of clients either on the drone side or in the U-Space services side. (H, H)
	Availability	Availability	Communication system should guarantee that it is up and working more than a specific amount of time per year. (H, H)
	Availability	Continuity	Communication system should guarantee that communication should be given continuity more than a specific amount of time per year. (H, H)
	Reliability	Integrity	The system must encrypt the information sent within the system and not allow unauthorized users to enter the system reach private information. (H, L)
	Reliability	Durability	The system should allow store information important data. (H, M)
	Security	Authentication	The communication system should only allow authenticated clients to interact with the system. (H, L)
	Security	Authorization	The communication system should limit the access within authenticated users. (M, L)
	Security	Encryption	The communication system should encrypt all communications. (H, L)
	Modifiability	Compatibility	The communication system should accommodate both a centralized and federated solution. (H, H)
	Configuration	Logging	The communication system should log important and relevant events. (H, L)

Table 5.1: Utility Tree

5.2.1 Simplified Concept

The communication system pursues a decoupled approach, relying on publisher-subscriber message queue mechanisms for supporting the communication flows between U-Space service components, which can easily accommodate asynchronous communication patterns. Such message queues are implemented by means of message broker service instances, which take care of handling queue operation semantics, also providing quality of service (QoS) guarantees.

It is assumed that the UAV will include a soft-transponder in its embedded software stack, in the form of a communications agent, which will provide a continuous and regular telemetry feed to the U-Space services, as shown in Figure 5.1.

Not all services will consume this information, with some services communicating with each other and consuming the output of other services (for a better understanding of how the U-Space services work, refer to Appendix B - BUBBLES services). The broker will manage the information so that clients do not need to keep track of other clients and can be appropriately decoupled.

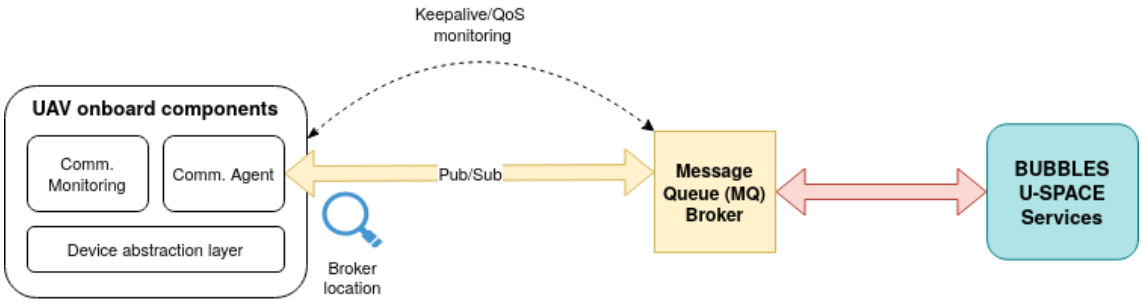

Figure 5.1: Simplified Architecture for the Communications Platform Concept.

The decoupling between communication entities is a crucial enabler in scalability and availability. This explanation is a simplified version of the architecture. In reality, the message broker will be composed of two separate message brokers, the edge broker, and the core broker.

Separating the Message Broker into two further helps in scalability and availability because those two brokers will have different functions, one related to drones the other one with the U-Space services. We can tailor each broker to their client's specific needs. Moreover, is the way the system will accommodate the two different architectural requirements.

5.2.2 Edge brokers

As shown in Figure 5.2, these brokers will be the first hop of the UAS telemetry. The Device abstraction layer will give access to the drone's flight information. Due to the nature of the U-Space project, multiple drones with different interfaces must communicate with U Space services. To handle the different interfaces is why the abstraction layer was created.

With the UAS flight information (made public by the abstraction layer), the communication agent is responsible for sending the information to the edge brokers.

Edge brokers are intended to be deployed in the perimeter of the telecommunications access network (for instance, co-located with cellular base stations). These brokers' primary

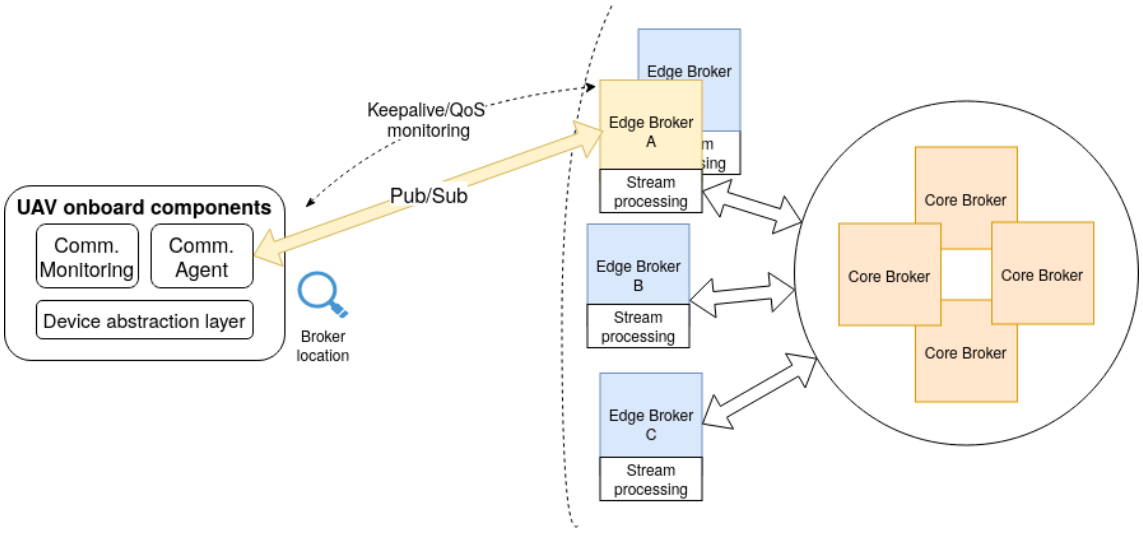

Figure 5.2: Edge Broker Architecture for the Communications Platform Concept.

objective is to forward the messages they receive to the core brokers, where the U-Spaces services will consume the information.

With the information from Section 2.2, the best fit for UAV-edge communications is the broker-less Data Distribution Service (DDS). DDS is a mature protocol that uses UDP and TCP transport to provide flexible, reliable, and low-latency message transport capabilities. DDS was chosen over the other alternatives, MQTT and MAVLink, due to its suitability to support a UAV communications profile, including native support for mechanisms for automatic endpoint location and granular QoS enforcement.

Its built-in automatic discovery for endpoint location capabilities perfectly fits UAS needs, helping the drone establish communications in edge situations (e.g., hand-over).

In addition, its many Quality of Service (QoS) policies enable a high-end level of customization, so it is possible to fine-tune the communication performance to the UAS mission requirements (stated by Methodology for the U-Space Safety Assessment (MEDUSA)). Finally, DDS security gives enough options to comply with the security requirements.

Since the Edge brokers are closer to UAS activity, they can respond faster in case of a problem. This reference architecture paves the way for the development of edge services hosted at the perimeter of the radio access network (within the radius of the first hop), which would allow a federated architecture.

5.2.3 Core brokers

Edge brokers will relay the messages to the core broker infrastructure, which is from U-Space Services will consume information. Core broker infrastructure was created to be the backbone of the U-Space services communications.

Figure 5.3 shows the core broker concept. The core brokers' primary purpose is to allow communications between the U-Space services and keep the information available even in catastrophic events (like a fire in a big database). Because of this, it needs to be designed to be redundant, geographically dispersed (across different data center points of presence), and resilient.

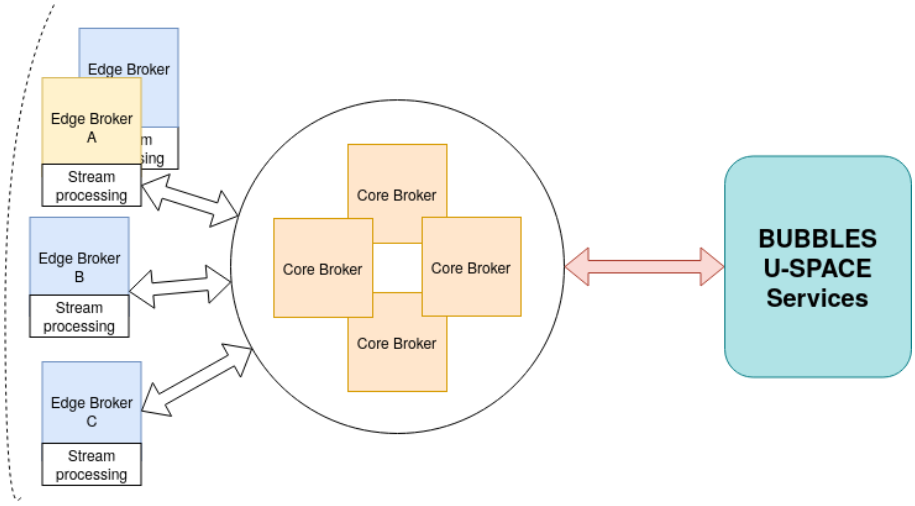

Figure 5.3: Core Broker Architecture for the Communications Platform Concept.

With the information from Section 2.3, the best fit for the core broker is Apache Kafka. Kafka was chosen over the other alternatives, AMQP and MQTT, due to its native scalability options, including topic and partition distribution and replication within the cluster, as well as its reliability options, including writing to disk and allowing clients to consume past messages.

Kafka's design focused on performance and persistence fits perfectly to core brokers' needs. Its use of the page cache enables vast amounts of traffic while still guaranteeing persistence to the disk.

This design, combined with data replicating across distinct geographical positions, makes it that information can be distributed across different data center points of presence, robust even against multi-site catastrophic incidents while keeping up with the performance the U-Space services needs. Finally, its security options are enough to comply with the security requirements.

5.2.4 Centralized Operations Reference Model

The centralized version will be implemented during the development part; it is the elected architecture in manned aviation and what BUBBLES favors.

Figure 5.4 shows the diagram with a centralized version of the architecture. In the centralized version, all communications start at the drone, where the DDS Publisher (communication agent) will periodically send the flight information to a DDS Topic.

The edge brokers, which include a DDS Subscriber subscribed to the before-mentioned topic, are then responsible for forwarding that information to a Kafka Partition via a Kafka Publisher. Kafka will then distribute that information according to the replicating factor chosen. On the other end, there will be some U-Space services consuming the UAS information (i.e., Tracking service and CNS Monitoring). Their output is sent to Kafka and consumed by other services, and so on.

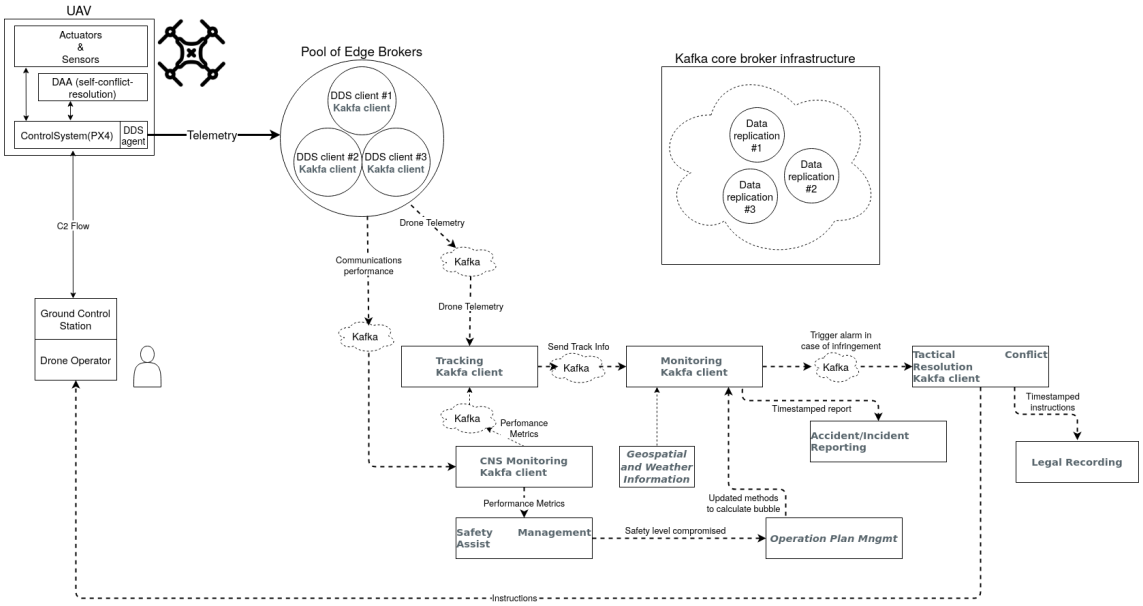

Figure 5.4: Centralized Architecture Reference Model.

5.2.5 Federated Operations Reference Model

One of the benefits of this solution is the adaptability to a federated solution. Figure 5.5 shows an architecture based on a federated approach.

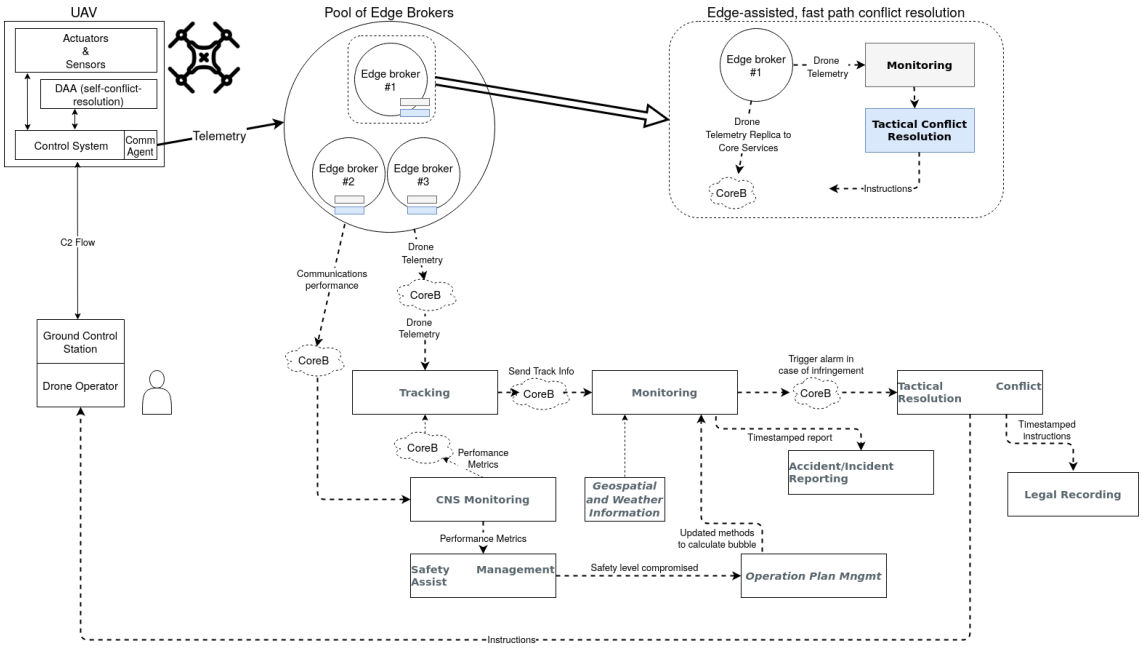

Figure 5.5: Federated Architecture Reference Model.

This approach takes advantage of the Edge Broker's proximity to the in-flight UAV communication units to implement geographically aware low-latency mechanisms. Stream processing capabilities at the edge of communications would allow different providers to host their services, such as tactical conflict resolution, which offers more successful responses closer to UAS activity.

The Core brokers would still receive a replica of the information, distribute it, and make it available for the different providers.

Apache Kafka offers a flexible deployment approach, which can start with a single-server and evolve up to different distributed topologies, with different arrangements for co-located nodes. This can be redundant storage rings and multiple messaging domains in a hub-spoke fashion, like shown in Figure 5.6. This is one of its biggest strengths, which can be leveraged in the scope of the core broker support infrastructure to comply with a federated approach.

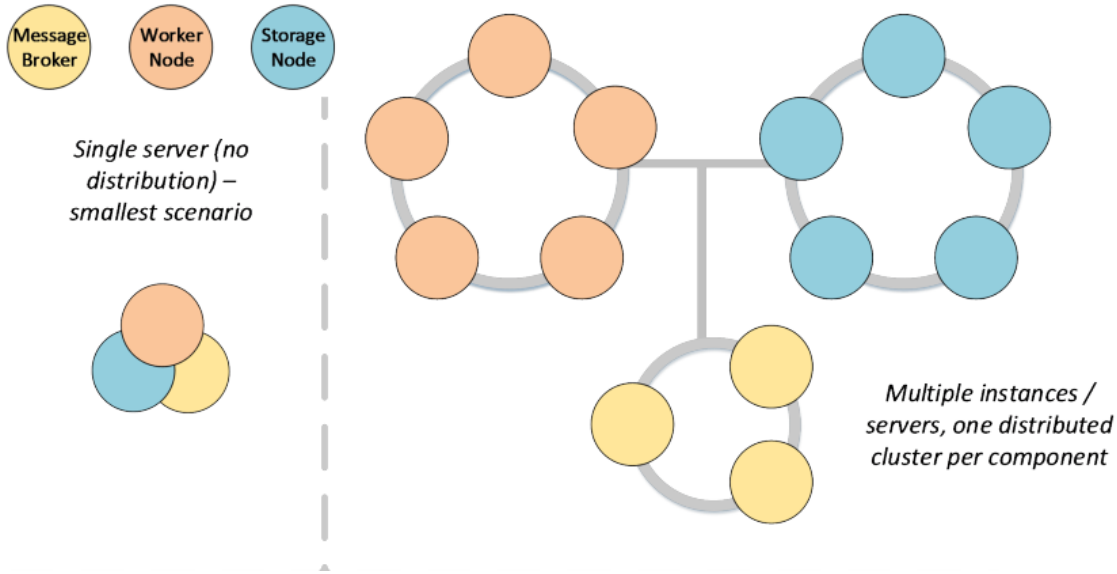

Figure 5.6: Examples of Kafka Topologies.

5.3 Summary

This chapter detailed two reference models for the project's architecture, both built from the Architectural drivers defined. The next chapter will delve into the development process details, which will implement the centralized model of the communication platform architecture.

This page is intentionally left blank.

Chapter 6

Development

This chapter is dedicated to solution development. It is the implementation of the architecture using the technologies picked.

It will start by presenting the implementations picked for each protocol and how the brokers and clients were configured, followed by how the code was built and organized. It will finish with how each component of the architecture was created.

6.1 Implementations

In this section, the implementations chosen for the protocols are presented.

For the DDS protocol, it was picked FastDDS. Apache Kafka offers its broker and client implementation; for this project, Apache Kafka's broker with librdkafka was used. For the Device Abstraction Layer, MAVSDK was picked.

The following subsections explain why they are picked and the configurations used.

6.1.1 FastDDS

FastDDS is a free and open-source C++ implementation of the Data Distribution Service (DDS) protocol by eProsima [104]. Both PX4 and ROS2 use FastDDS to allow off-board devices to behave as onboard components. In addition to this, it is well documented. For those reasons, it was picked as the implementation for supporting DDS in the proposed communication architecture.

DDS is data-centric, so a definition of what the data will be and what format is required. FastDDS uses this requirement to facilitate development with the use of the fastddsgen tool. The tool takes IDL files as input (C++ struct file) and creates the necessary classes for the functioning of both Publisher and Consumer. This includes write and read methods, initialization and configuration for both types of client, and marshaling and unmarshalling methods based on the IDL structure.

Table 6.1 shows the drone telemetry information used by Indra in their project. We based the solution on that and created IDL files accordingly.

After running the fastddsgen tool with the idl file created, the only thing left to configure is the security and Quality of Service (QoS) policies from the options stated in Security

Attribute	Description	Attribute Type	Data specific
longitude	longitude coordinate	float	degree
latitude	latitude coordinate	float	degree
altitude	altitude	float	meters
yaw	yaw angle	float	degree
row	row angle	float	degree
pitch	pitch angle	float	degree
gs	ground speed	float	m/s
track_angle	track angle	float	degree
vert_speed	vertical speed	float	m/s
battery	remaining battery	float	%
time	time for telemetry plot	uint64	ms UNIX format

Table 6.1: FastDDS data model.

and Quality of Service stated in Chapter 2 (subsection 2.3.2).

Security configuration

Since FastDDS follows the DDS protocol requirements, it has access to the built-in plugins. For the development of the communications system, four plugins will be used, and they are the authentication plugin (DDS:Auth:PKI-DH), the access control plugin (DDS:Access:Permissions), and the cryptographic plugin (DDS:Crypto:AES-GCM-GMAC).

For the authentication plugin, first, a self-signed X.509 certificate was created to function as Certificate Authority (CA). After that, different X.509 certificates signed by the CA were attributed to different communication clients. Only clients holding certificates signed by the CA will be able to communicate.

The access control plugin works in tandem with the authentication plugin. With information based on the client's certificate, DDS will distinguish what topics and privileges the user has. DDS publishers will only be able to write to topics and edge brokers to read them.

The cryptographic plugin handles all encryption and decryption, and it was configured to encrypt all messages.

QoS configuration

From the QoS files, two were configured, the RELIABILITY_QOS and the DURABILITY_QOS.

The RELIABILITY_QOS was configured on both the DataReaders and DataWriters to offers at-least-once delivery (RELIABLE). For the communications to be reliable, the HISTORY_QOS needed to be configured on the DataWriter.

The selected type was KEEP_ALL_HISTORY_QOS. This way, all messages sent by the DataWriter are saved, and in case of error, due to the RELIABILITY_QOS being configured, it will re-send saved messages that the subscriber could not access.

6.1.2 Apache Kafka and librdkafka

Apache Kafka has its protocol implementation, using java and scala, the first one for the clients and the second one for the brokers. Since Fast DDS is c++, Apache's brokers were used with Kafka c++ implementation, librdkafka, to write the subscriber and publisher. This way, integrating both the DDS Subscriber and Kafka Publisher on the Edge broker is easier.

In addition, the creators of librdkafka are Confluent, which was founded by the original Kafka developers. For those reasons, it was picked as the implementation for Kafka Clients.

Security configuration

For authentication of the client and encryption of the client-broker channel, the broker was configured to use TLS. TLS uses certificates, so a self-signed X.509 certificate to function as CA, and various X.509 certificates signed by the CA are distributed between clients.

For broker and Zookeeper authentication and encryption of their channel, Zookeeper was configured to use TLS.

In addition, access control by ACL's was configured, also using information about the certificates. The only restriction was that certificates belonging to edge brokers will only be enabled to write.

QoS configuration

In terms of QoS, for Publishers and Subscribers, the default configuration was the one used, which is at least once. Publishers will expect all ACKs from the broker, and Consumers will process the message before committing the offset to the broker.

6.1.3 MAVSDK

In a real scenario, the drone would need a Raspberry pi, or a similar type of equipment, attached to the flight controller to have access to flight information, using serial communications (e.g., UART).

MAVlink is a well-known drone communication protocol that supports TCP, UDP, and UART and is compatible with most autopilots, including PX4 and Ardupilot, and Ground Control Stations. MAVSDK is an implementation of MAVLink supported by the DroneCode project. It offers compatibility with C++, Python, IoS, and Android. The MAVLink c++ implementation was chosen to facilitate integration, as well as being the oldest of the bunch and offering Heartbeat connections, which allows C++ implementations to initiate connections, in contrast with the Python implementation.

Because of those reasons, MAVSDK C++ was picked to develop the component responsible for getting drone telemetry; the only configuration needed is the URL to the autopilot, which will be used to initiate a connection and tell the autopilot to send information to this add-on.

6.2 Build and execution of the service

This section shows how the project was set up to build the code and configure it.

The main idea behind the code organization is explained in the following subsection, followed by the explanation of each communication service component and how it was configured.

6.2.1 Main Concept

FastDDS's tool fastrtpsgen generates five classes and four header files. One of the classes acts as the main. For that reason, it was discarded.

One of the classes is BubblesTelemetry, which creates the methods and structs necessary to create an object with the required parameters in the IDL file used to generate.

It also creates a BubblesPublisher and a BubblesSubscriber class to make the publisher and subscriber, respectively. Both these classes have a listener subclass. The publisher listener has a method for listening to matches to subscribers. The subscriber listener has two methods, one listening for matches to publishers and another one listening to new messages.

The last class is BubblesPubSubTypes which creates the methods for serialization and deserialization of information between DDS entities.

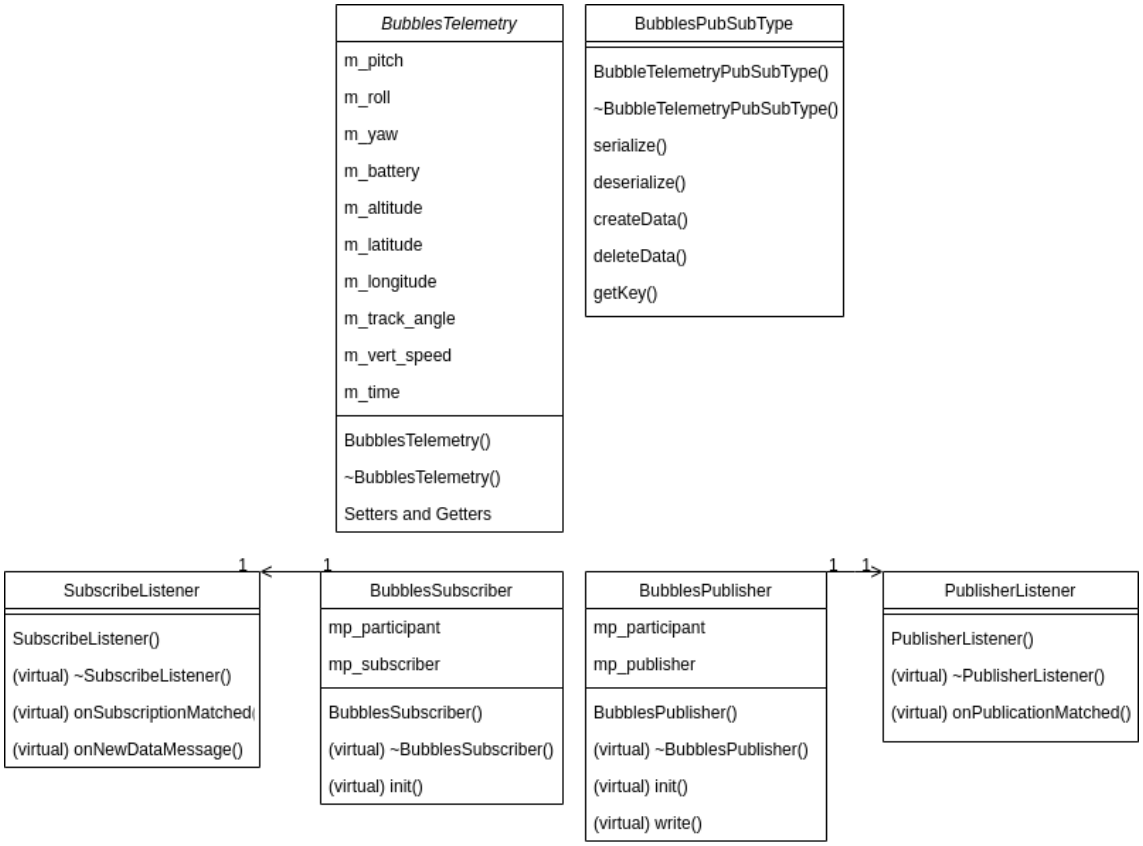

Figure 6.1: DDS classes diagram.

Only minor adjustments had to be made. The security and QoS default configuration were added, and the ability to read configurations from a file. The original headers and classes for the Subscriber and Publisher were also changed. Like the write() and onSubscription-Matched() methods for each class, specific methods were configured to be virtual, and class variables were set to protected to allow access from Derived classes.

C++ virtual methods allow polymorphic relationships. This means that Derived classes (children) can implement the method, and when running the code, the most derived method will be executed, which will be the children method instead of the base class (parent). Virtual methods and abstract methods are not the same because virtual methods need to be implemented in the base class so that derived classes can execute the default methods if they do not define one.

Virtual methods grant the personalization of each component; they can change the methods required for their task while all components are reading from the same source.

The Kafka source code was created with the librdkafka implementation. Two classes and header files were created—one for the consuming part and the other for the publishing part. Both have a init method to initialize the objects and configurations. The Security configurations were added to the broker properties, and QoS and additional Security default configurations were added on each class. Both classes have the ability to read configuration from a file, similar to the DDS classes.

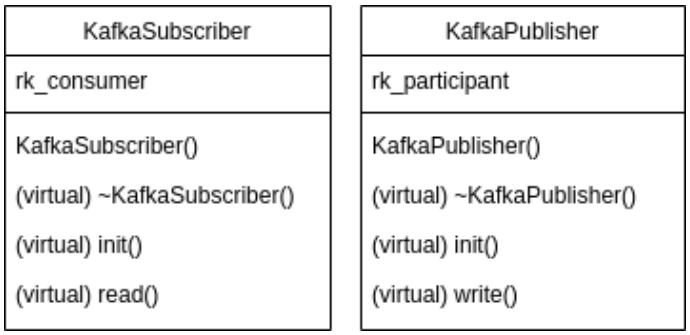

Figure 6.2: Kafka classes diagram.

In addition, serialization and deserialization methods had to be created. Librdkafka Publisher "send" methods accept a string as an input. So a serialization method to transform the BubblesTelemetry object to a string was needed. The method created separates each section of the string by a character (";"), and each section is composed of a short name (parameter's short name) and its value delimited by another character ("="). The deserialization method returns the BubblesTelemetry object by dividing the string with the appropriate delimiters.

For MAVSDK, only one class was created because the communication system only needs to read flight information from the drones. Follows the sample principle, a init method to initiate the objects are to configure the system, and a read method to get values from the drone's flight. It was also changed to accept configurations from a file, but the only configuration needed is the URL of the autopilot.

In order to facilitate the building and compiling of the sources, CMake was used. Cmake is an open-source, cross-platform family of tools designed to build and package software. Kitware created Cmake when it noticed the lack of options for a cross-platform build environment for open-source projects.

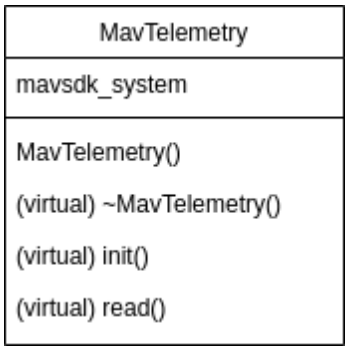

Figure 6.3: MAVSDK class diagram.

It is used to control the software compilation process using simple platform and compiler independent configuration files and generate native makefiles and workspaces that can be used in the compiler environment of the user's choice.

This arrangement makes it that each component can be incorporated into each other with minimal effort, especially with CMake creating the appropriate makefile for each case and environment.

By changing the config file (init methods accept the file as input), the clients can communicate to different topics, different brokers with the same source code, or choose other certificate files.

For components to use the source code, they need a file with the main function. This file can define the derived class(es) and override the virtual method(s). This way is also easier to integrate, for example, a Kafka Publisher with a DDS Subscriber or a DDS Publisher with a Kafka Subscriber. Each file can define the read and write methods for each class. Each component is explained in the following subsections.

6.2.2 Communication agent

The communication agent is responsible for sending drone telemetry to the edge broker. It will need two components, the layer that gets Unmanned Aerial Systems (UAS) flight information using MAVSDK and the layer that sends the information to the edge broker using FastDDS. Figure 6.4 shows the communication agent diagram.

For the flight information component, MAVSDK needs the URL for the UAS. This can be a serial port, UDP, or TCP connection. That information is retrieved from a configuration file. After a connection, MAVSDK will retrieve drone telemetry periodically (the default is each second, but it can be changed), and it will send information using a FastDDS publisher.

Since no particular changes need to be done to the publisher, the BubblesPublisher class was used. The class has two public methods, init() and write(). The DroneTelemetry init method initiates the BubblesPublisher object, and the read function was changed to call the write() method for each time it receives information from the drone.

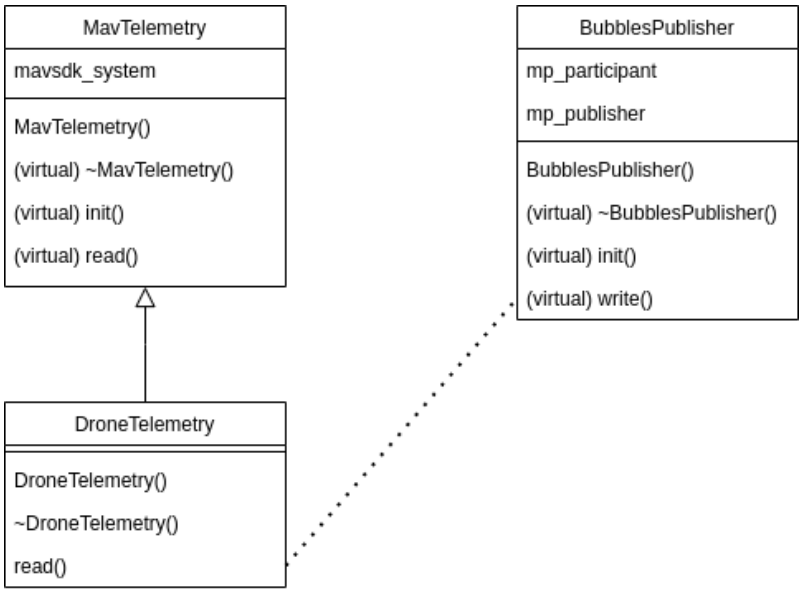

Figure 6.4: Agent classes diagram.

6.2.3 Edge brokers

Edge brokers must receive information from the Communication Agent and will send it to the Core Brokers. The component responsible for receiving information is a DDS Subscriber and, the component responsible for sending is a Kafka Publisher. Figure X shows to diagram concept for one edge broker in the centralized version.

Similar to the Communication agent, no particular change to the publisher is needed; as such, the KafkaPublisher class was used. The EdgeDDS class was created to derive from the BubblesSubscriber, and EdgeListener classes were created to override the onNewDataMessage method.

EdgeListener class is responsible for initializing the Kafka object and use its write method to send messages to the Kafka Broker for each message the DDS subscriber receives.

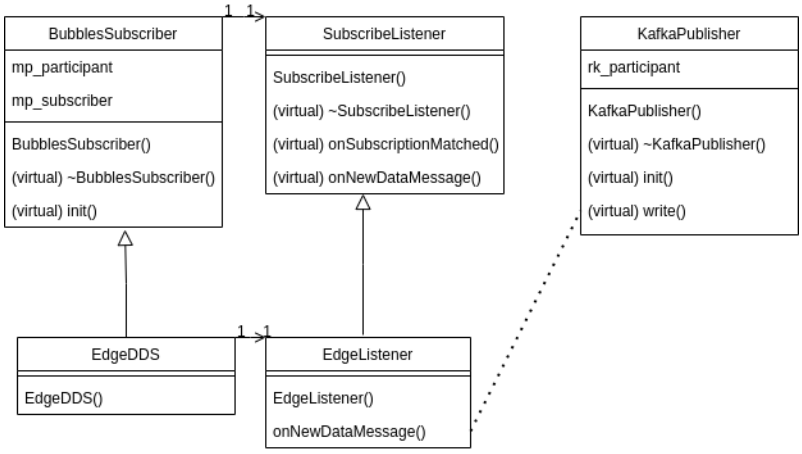

Figure 6.5: Edge classes diagram.

6.2.4 Core brokers

The Core infrastructure only uses Kafka; therefore, Kafka Publishers send information to the brokers' partition, and Kafka Consumers will get their messages from the brokers.

Ultimately USpace Services will be the final consumers of this information. The services that listen to Flight Telemetry will be consuming the information pushed by the Kafka Publishers located at the Edge Brokers. After processing that information, it will publish its output to Kafka so that other services can process that information.

Only the class CoreKafka deriving from KafkaSubscriber was needed to be created. The publisher only requires changes to the configuration file, and as such, the KafkaPublisher class was used.

This component is similar to the Edge Broker, but only with a class derived from the KafkaConsumer instead of BubblesPublisher. CoreKafka initializes the KafkaPublisher object and publishes information received using the write method.

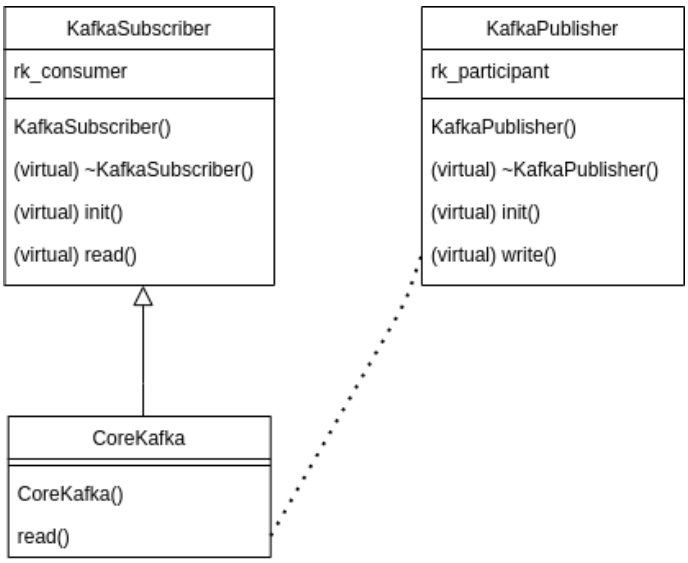

Figure 6.6: Core classes diagram.

6.3 Summary

This chapter detailed how the code was built and the configuration for each component of the architecture. The next chapter will delve into the details of the testing process, which will present the test environment and test scenarios created to test the components developed.

This page is intentionally left blank.

Chapter 7

Testing

This chapter will be dedicated to the testing environment, scenarios, and results. It will start by showing the testing architecture and explain the tools used, netEm and iPerf, to shape traffic and verify the shaping. Next, the different testing scenarios picked, defining each one and, in the end, the result of each test scenario.

7.1 Testing Environment Architecture

In order to validate the communication solution, a test environment needed to be developed.

To simulate drones, we will be using PX4. Although ArduPilot has better hardware compatibility, as stated in subsection 2.4.3, PX4 offers much better accuracy, resulting in better flight information consumed by the U-Space services.

PX4, with the help of a simulator, will act as the simulated drone. The flight telemetry information feed will be acquired from the autopilot.

The simulator recommended by PX4 is Gazebo, an open-source 3D robotics simulator software. Gazebo is a widely used simulator for robotic vehicles in general, allowing the accurate reproduction of environments (scenarios) that can be traversed by robots, consisting of UAVs in our case. Gazebo excels at portraying complex bodies through a combination of simple geometric shapes and joints. These joints connect the shapes together, allowing them to perform dynamic motions.

Each drone will receive commands from a QGroundControl (QGC) station. QGC is also under the governance of the Dronecode Project, like MAVLink. It is an open-source mission planner with an automated graphical mission planning interface. It allows operators to plan missions automatically, upload said missions to the UAS, and perform a replanning during the execution. Finally, while the mission is being executed, the UAV can be monitored in the 2D image provided.

As shown in Figure 7.1, the communication system will get the PX4 flight information and then supply it to the U-Space Services to understand how well the communication system performs. Unfortunately, the BUBBLES work got delayed, and the U-Space services were not available to test the communication system, not even Indra's tracker. The focus was on testing the flight information consumption by the U-Space Services, so only a Kafka Consumer was used on the U-Space services side.

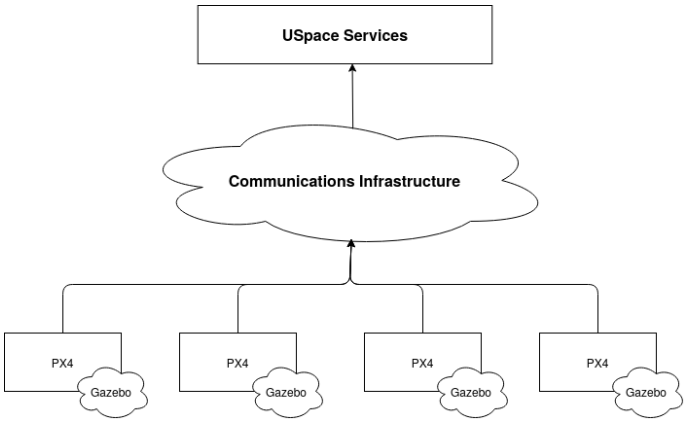

Figure 7.1: Test Architecture.

7.1.1 NetEM

NetEm [105] is a network link emulator capable of shaping communication flows, accordingly to specific criteria. It is an enhancement of the Linux traffic control facilities that allows the kernel to shape the network parameters for egress (outgoing) packet flows. from a selected network interface. As shown in Figure 7.2, it is configured to behave as a transparent bridge. This will provide the means to constrain/disturb the traffic in a transparent way so that both sides of the communication will not be aware of its existence.

Network link emulation can take into account several different scenarios, namely: specific Upload/Download rates, jitter, packet loss, and bit-error-rate (BER) disturbance and latency. Together with the native Hierarchical Token Bucket traffic shaping capability incorporated in the Linux kernel (which allows for stochastically fair bandwidth sharing), the netEm network emulator will allow configuring specific network profile test parameters independently for the uplink and downlink directions.

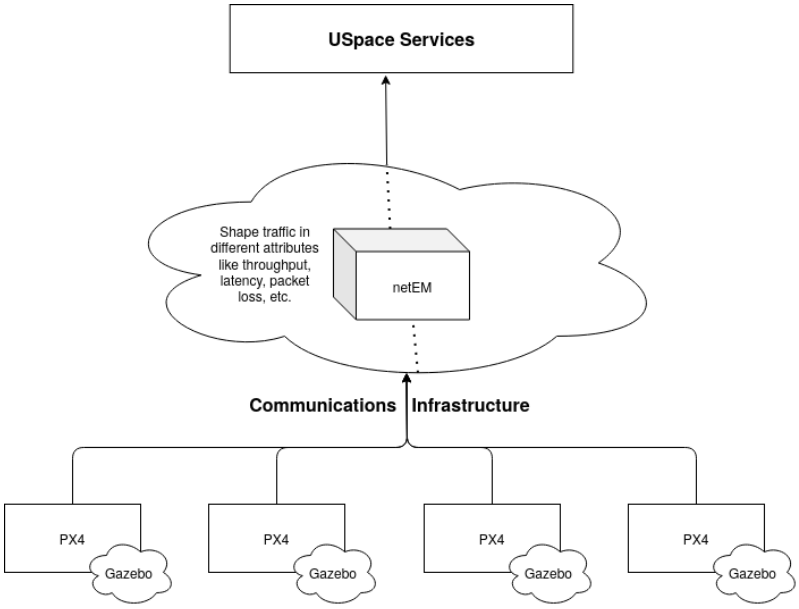

Figure 7.2: Test Architecture with netEm.

7.1.2 iPerf

Perf3 [106] is a network performance and diagnostic tool, which provides the means to generate TCP, UDP and STCP traffic streams between two endpoints: a server and a client instance. This tool allows the user to customize a series of connection parameters, such as the TCP window/buffer size, packet size, or the UDP stream bandwidth, providing insights about jitter, packet loss and bandwidth performance indicators across the two testing endpoints.

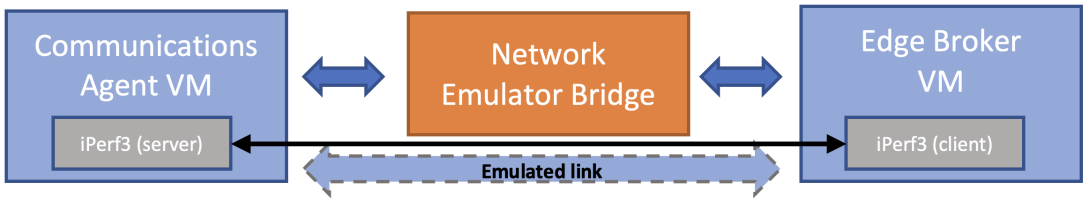

Figure 7.3: Iperf use diagram.

In the scope of the effort hereby documented, iPerf3 was used to tune and validate the enforcement of the bandwidth, packet loss and latency parameters configured in the network emulator bridge. Bidirectional tests were undertaken to confirm that the enforced shaping policies were correctly deployed in both directions of the connection. For this purpose, an iPerf3 instance was executed on a host deployed on one end of the transparent network emulator bridge while another instance was executed on the other end. TCP and UDP tests were undertaken to verify the behavior of the emulated link.

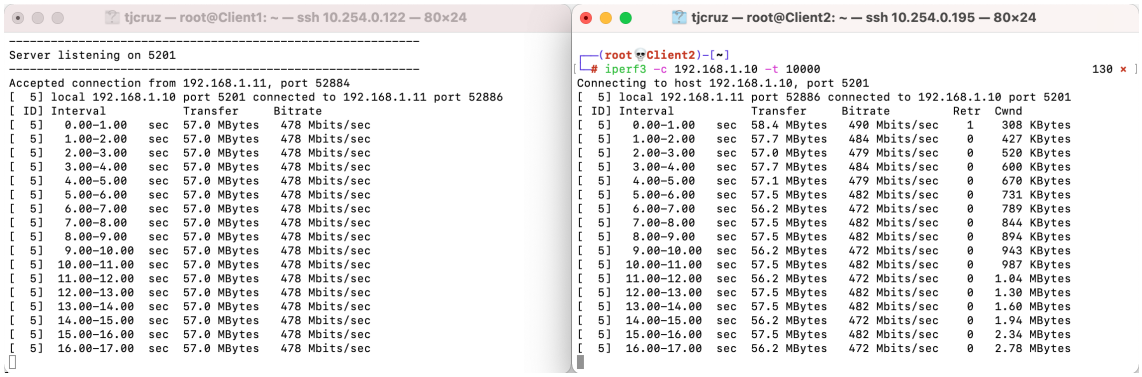

Figure 7.4: Iperf test with no limitation.

7.2 Testbed

The setup was, as shown by Figure 7.7, composed of three Virtual Machines (VMs). The first one was for the PX4 environment plus the communication agent component to successfully send information to the edge broker. The second virtual machine is for the edge broker, which will publish all the information to a Kafka broker running on the third virtual machine. Also, the third virtual machine is running the Zookeeper instance and the Kafka Consumer to simulate the U-Space services consumption.

Between the drones and the communication service, netEM is deployed. A VM hosting the

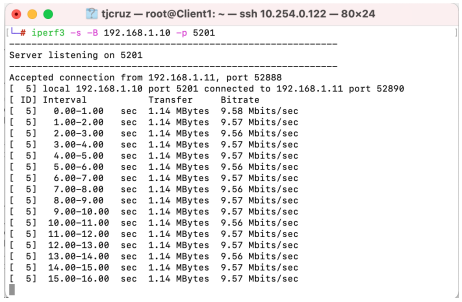

Figure 7.5: Iperf test with 10Mbit/s UL/DL.

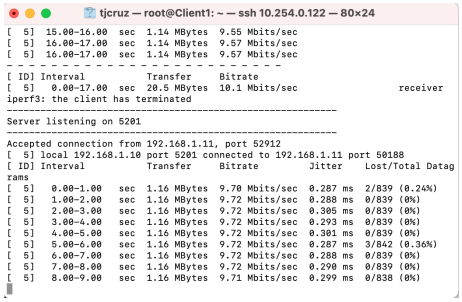

Figure 7.6: Iperf test with 10Mbit/s, delay 1ms and loss 0.1%.

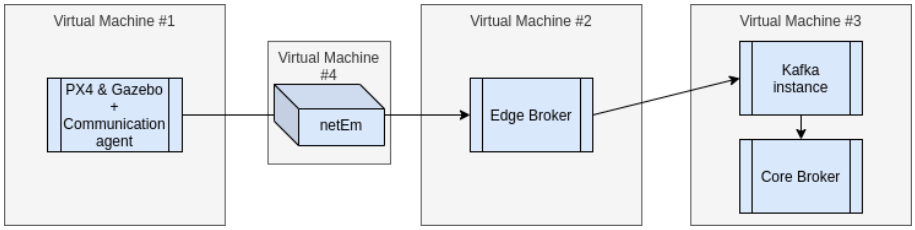

Figure 7.7: Testbed diagram.

network emulation tool, fine-tuned for a real-time profile, with locked CPU and memory reservations, acts as a transparent layer two bridge between the two ends.

This is achieved by creating two port groups within a Virtual Switch. Then, both VM #1 and #2 were added to port groups 1 and 2, respectively. The netEM virtual machine was then added to both port groups.

Since the only network connecting VM#1 and #2 is through the netEm VM, packets have no other choice than to pass through the transparent bridge, which will be altering communications as the configurations defined.

All the VMs, with the exception of the netEm, which does not need so much power, have the following specifications:

- 48GB hard disk
- 8GB RAM
- Intel(R) Xeon E5-2620 @ 2.10GHz - 8 vCPUs
- Created using Hypervisor VMware ESXi 7.0

7.3 Testing Scenarios

There are four main test categories, Scalability testing, Latency testing, Abnormal Condition testing, and Functional requirements testing.

Scalability tests are run to measure the throughput of the communication service for capacity estimation. In these tests, a DDS Publisher, acting as the drone, produces an unlimited number of messages to the edge broker.

The throughput is the number of messages consumed by the KafkaConsumer on the U-Space service side, counted for each test. No type of reliability was configured for the DDS Publisher for these tests as it would only increase latency due to so many lost packets.

The throughput tests will be divided into two sections: tests with one topic and with two topics. With one topic, 1, 5, 10, or 20 DDS Publishers all producing an unlimited number of drones to one Edge broker, listening to one DDS topic, and sending the messages received to one Kafka broker. Finally, a Kafka Consumer will be reading these messages and counting how many it reads.

With two topics, 1, 5, 10, or 20 DDS Publishers will be producing messages, but per topic. Two Edge brokers will be running, each listening to one topic. Both will be sending messages to the same Kafka topic, which two Kafka brokers will share. In the end, will be two Kafka consumers of the same consumer group, consuming the messages sent by the Edge brokers and counting them.

When only one broker is used, only one partition is used for the topics. Two partitions are created when two brokers are being used so that brokers are responsible for different partitions. The partitions are also configured to set its replications rate to two, in the two-topic scenario, as setting the replication rate with one broker does nothing. Since Edge brokers communicate to random partitions when two brokers are used, they should be used equally.

Latency tests are focused on round trip time (RTT) latency measurement for message propagation. By measuring the timestamp when the drone sends the message and comparing it to the timestamp when it receives the message, the RTT latency is measured. The latency has to be measured in RTT time because, with the test environment, VMs, synchronizing the clocks at the granularity of milliseconds would be extremely hard.

Because of this, extra components had to be developed. The reverse of the Edge broker, where a KafkaSubscriber is listening, and DDSPublisher sends messages had to be created. The core broker also had to implement a KafkaPublisher, to forward the message the other way. Finally, the drone had to implement a DDS Subscriber responsible for receiving information from the reverse edge component created.

In addition to UAV to Core RTT latency measurement, tests for UAV - Edge RTT latency were created. These UAV-Edge tests required another component, which receives DDS messages and sends them, also via DDS, to the drone via the returning topic.

Latency tests will test 1, 5, 10, or 20 Drones, all producing a controlled number of messages via DDS topics to one Edge broker. The edge broker is then sending the messages received to one Kafka broker. On the other end, the Kafka Consumer is returning messages to another Kafka topic. A reverse of the edge broker is listening and sending those messages to the drone again to compare the timestamps and save the difference.

Latency tests will be using a controlled message rate gathered from the preliminary tests executed. The frequency picked was 50 Hz, so that when 20 drones are used, the messages are produced a little bit above the maximum throughput of the communication platform.

Abnormal conditions testing measures both the message throughput and RTT latency when the communication's performance is being altered by netEM. These tests measure both throughput and latency under non-nominal conditions, and this goes in line with the MEDUSA methodology.

For bandwidth, the criteria for choosing values were based on an intersection of the common ranges found across popular technologies, together with the feedback obtained from

the interactive explorative process, which is implicit in the test procedures. For instance, certain bandwidth values were entirely irrelevant for some tests due to the fact that the QoS level in use and the number of drones were not even close to exhausting them. The preliminary tests showed that bandwidths bigger than 250 Kbit/s did not alter communications, so the range of bandwidths picked, in Kbit per second (Kbit/s), were 25, 50, 100, and 250.

Latency is measured in RTT, so alterations done by netEm in terms of latency will affect the message on both trips. Some of the values found from the literature review on the network technologies were unrealistic for a premature communication platform.

RTT Latency values were considered in the same range as throughput. The values picked, in milliseconds (ms), were 10, 30, 50, and 100.

Packet loss was considered based on the values found on the literature review, 0.5 %, 1 %, and 5 %, and added two terrible conditions with 10 % and 20% packet loss.

Tests for abnormal communications conditions will be performed for a single drone instance, producing a controlled number of messages via DDS topics to one Edge broker. The Edge broker is then sending the messages received to one Kafka broker. On the other end, the Kafka Consumer is counting those messages and returning them to another Kafka topic. A reverse of the edge broker is listening and sending those messages to the drone again to compare the timestamps and save the difference.

Functional requirement tests were undertaken to verify that the functional requirements were successfully implemented. These tests will go through each functional requirement one by one and verify that it is implemented.

Each configuration is run for 10 minutes for all types of tests and is repeated five times to guarantee statistically relevant data. In addition, to guarantee the same conditions, all components are reset, and queues are cleared after each test. Table 7.1 shows the summary for the test scenarios values.

<table>
<tr><th>Scalability Tests</th><th>UAV instances</th><th>Broker Mix</th></tr>
<tr><td>1 topic</td><td>1,5,10,20</td><td>1 Edge, 1 Core</td></tr>
<tr><td>2 topics</td><td>1,5,10,20 (per topic)</td><td>2 Edge, 2 Core (1 topic each)</td></tr>
<tr><th>RTT latency</th><th>UAV instances</th><th>Broker Mix</th></tr>
<tr><td>UAV to edge</td><td rowspan="2">1,5,10,20</td><td rowspan="2">1 Edge, 1 Core</td></tr>
<tr><td>UAV to core</td></tr>
<tr><th>Abnormal Conditions</th><th>Values</th><th>Scenario</th></tr>
<tr><td>Packet Loss (%)</td><td>0.5, 1, 5, 10, 20</td><td rowspan="3">1 UAV, 1 Edge, 1 Core</td></tr>
<tr><td>RTT Latency (ms)</td><td>10, 30, 50, 100</td></tr>
<tr><td>Bandwidth (Kbit/s)</td><td>25, 50, 100, 250</td></tr>
</table>

Table 7.1: Test scenarios.

7.4 Tests Results

This section will present the test results and analyze them. The following subsections will show the scalability tests, the RTT latency tests, the Abnormal condition tests, and the Functional tests.

7.4.1 Scalability tests

Scalability tests show how the communication platform scales, and as such, messages sent were not be limited to see the maximum throughput of the system.

Table 7.2 shows the number of messages consumed by the Kafka Consumer (U-Space Services) for each group of five tests. These tests are with one edge broker and one core broker (and Kafka Consumer).

# Drones	Mean	SD	Max	Min
1 Drone	553,515	51,718	630,690	523,965
5 Drones	492,655	1,191	494,684	491,574
10 Drones	491,354	2,264	494,005	488,412
20 Drones	491,869	3,849	494,843	485,900

Table 7.2: Total number of messages consumed by the Kafka Consumer with 1 Topic, during 10 minute runs.

The results show that as the number of drones increases, the number of messages consumed goes down. They also show that the Standard Deviation (SD) increases with the number of drones. There is one outlier in the SD values, the one drone in the one topic scenario, where one of the test runs had a much bigger number of messages consumed with over six hundred thousand messages. That is why its SD is 51,718, much higher than the rest of the tests.

Having more drones means that the edge brokers need to handle more clients sending the same number of messages. Also, only one VM was responsible for creating all the different communication agents' processes. The time the Operative System needs to schedule what process will send its messages increases as more processes share CPU and Memory, which might result in small losses of the number of messages throughout the 10-minute run.

Table 7.3 shows the same values for the test with two edge brokers and two core brokers (and Kafka Consumers). So the first scenario has two drones, one for each Edge Broker Topic.

# Drones (per topic)	Mean	SD	Max	Min
1 Drone	1,015,740	1,997	1,017,952	1,012,501
5 Drones	728,648	2,862	733,569	726,738
10 Drones	679,852	7,810	689,459	671,463
20 Drones	647,854	5,664	657,179	642,993

Table 7.3: Total number of messages consumed by the Kafka Consumers with 2 Topics, during 10 minute runs.

From the values with one drone for each topic, we can see that the replication factor of 2 nor the number of partitions affect the number of messages consumed significantly. With just one drone with one topic vs. two drones with two topics, the latter has almost double the message consumption while maintaining appropriate SD values.

However, the ratio between the 1-topic scenario and 2-topics scenario drops as more drones start to connect to the Edge brokers. The Communication Platform reaches around 650,000

messages with two topics and twenty drones, compared with the 490,000 with one topic and twenty drones.

In both tests (1 and 2 topics), the values in the 10-drone and 20-drone scenarios are similar, and the values for 5-drone scenarios only have a more significant difference in the two-topic tests, which seems to show that the VM behaves similarly when more than five processes are running. This can be attributed to the process scheduling problem.

Although one drone sends more messages than the maximum throughput, the number of messages still consumed is this high because the reliability for drone messages was turned off. Also, DDS uses UDP to send packets, so once they are lost, they are lost for good.

7.4.2 RTT Latency tests

Latency tests were made with a limited number of messages per second to understand how the system would behave under normal conditions. These tests will be done with one edge and one core, with 1, 5, 10, and 20 drones.

The number of messages per second was picked considering the maximum throughput of messages. Fifty messages per second (50Hz) results in about six hundred thousand messages being sent over the 10-minute run, which is a bit higher than the maximum throughput for the one edge one broker scalability tests results.

All the runs started from an empty queue, and a big spike at the beginning of the run was constant in tests. This is due to Kafka working poorly with empty queues. The first 150 latency values were removed to negate the early spikes, which are about 3 seconds of messages from the start of the run.

Table 7.4 shows the RTT latency values from each scenario for UAV to Core communications. For 1, 5, and 10 drones, the mean RTT latency is less than 2 ms, with the first two having max values lower than 100 ms and the ten-drone scenario reaching values of half a second. Although the max increases by a lot in the ten-drone scenario, the standard deviation shows consistency in the latencies measured.

# Drones	Mean (ms)	SD (ms)	Max (ms)	Min (ms)
1 Drone	1.3	0.5	38	1
5 Drones	1.3	0.5	24	1
10 Drones	1.6	2.3	505	1
20 Drones	55.5	53.9	698	8

Table 7.4: RTT Latency (ms), UAV to Core (10 minute runs).

The twenty-drone scenario has significantly more latency, but that was expected to be, as the throughput of 20 drones is over the limit of the communication platform. Still, a mean of 55.5 ms RTT latency is not bad for a platform over its limit.

Table 7.5 shows the RTT latency values from each scenario for UAV to Edge communications. The UAV to Edge latency tests show latencies lower than 0.5 ms RTT latency, even in the twenty-drone scenario. As the number of drones increased, the number of outliers also increased. There is a clear jump from ten drones to twenty drones in terms of outliers, from 46 ms to 353 ms.

However, the standard deviation does not increase much comparing the one-drone and the

# Drones	Mean (ms)	SD (ms)	Max (ms)	Min (ms)
1 Drone	0.2	0.4	12	0
5 Drones	0.3	0.5	34	0
10 Drones	0.3	0.5	46	0
20 Drones	0.3	1.5	353	0

Table 7.5: RTT Latency (ms), UAV to Edge (10 minute runs).

twenty-drone scenarios, which shows the consistency of the latency values from drone to edge.

7.4.3 Abnormal Conditions tests

Table 7.6 shows the number of messages consumed by the Kafka Consumer, with the different bandwidth values imposed by netEm. Table 7.7 shows the RTT latencies for the same group of tests as Table 7.6.

Bandwidth	Mean	SD	Max	Min
25 Kbit/s	1,337	643	2,464	849
50 Kbit/s	1,703	91.7	1,791	1,599
100 Kbit/s	1,873	139	1,997	1,642
250 Kbit/s	30,050	60	30,123	29,971

Table 7.6: Total number of messages consumed by the Kafka Consumer, with different bandwidth limits, during 10 minute runs.

The Bandwidth tests went as expected. For bandwidths lower than 250 Kbit/s, the communication platform cannot function, as the retries from the DDS publisher for the missing packets fill the queues. The number of messages consumed by the U-Space services is extremely low for the 25, 50, and 100 Kbit/s scenarios. Although the drones produce about 100 Kbit/s, the necessary additional traffic for reliability guarantees also fills the channel. That is why only the 250 Kbit/s scenario gives good results.

Bandwidth	Mean (ms)	SD (ms)	Max (ms)	Min (ms)
25 Kbit/s	404,550	109,553	517,965	265,854
50 Kbit/s	288,955	132,621	517,915	273,990
100 Kbit/s	111,942	55,228	484,293	104,977
250 Kbit/s	11.5	0.6	27	11

Table 7.7: RTT Latency (ms), with different bandwidth limits (10 minute runs).

The RTT latency value between consecutive packets constantly increases for the duration of the run, as more messages are produced than consumed. As for 250 Kbit/s is more than enough, having a mean of 11.5 ms RTT Latency from UAV to Core.

The next tests are the Additional RTT Latency tests, to see the throughput and latency when additional latency is being added up by netEM. Table 7.8 shows the number of messages consumed by the U-Space services, and Table 7.9 shows the RTT latency measured.

RTT Latency	Mean	SD	Max	Min
10 ms	30,010	28.8	30,036	29,991
30 ms	30,023	38.5	30,099	29,970
50 ms	30,022	26.6	30,043	29,991
100 ms	90,052	56.2	30,114	29,996

Table 7.8: Total number of messages consumed by the Kafka Consumer, with different emulated RTT Latency values, during 10 minute runs.

RTT Latency tests also went as expected. The latency did not affect the number of messages consumed by the Kafka Consumer, as the mean and standard deviation did not vary as more latency was added.

RTT Latency	Mean (ms)	SD (ms)	Max (ms)	Min (ms)
10 ms	21.5	0.6	41	21
30 ms	41.5	0.6	62	41
50 ms	61.5	0.6	77	61
100 ms	111.5	0.6	131	111

Table 7.9: RTT Latency (ms), with different emulated RTT Latency values (10 minute runs).

As for Uav to Core RTT latency values, it also was as expected; the mean increases as netEM increases the latency in communications. However, the standard deviation keeps steady in all tests (0.6 ms).

Packet loss tests divide into two categories, with and without additional latency. Table 7.10 shows the number of messages consumed by the Kafka Consumer, and Table 7.11 shows the RTT latency. Both for the packet loss without additional latency tests.

Packet Loss	Mean	SD	Max	Min
0.5 %	29,837	38.5	29,905	29,814
1 %	29,646	137	29,797	29,458
5 %	27,921	452	28,462	27,311
10 %	28,802	480	26,192	25,178
20 %	18,028	1,734	20,193	15,722

Table 7.10: Total number of messages consumed by the Kafka Consumer, with different emulated packet loss values, during 10 minute runs.

The number of messages consumed with different values for packet loss went as expected. As the loss percentage increases, the number of messages consumed decreases, and the standard deviation increases, as it is more random how many messages the consumer receives.

As for latency, it also went as expected; from 0% to 5%, the communication platform performed well, in the range of 10 to 20 ms RTT latency. As for 10%, it performed better than expected, being that an abnormal value by itself. However, it still managed to deliver messages under 40ms RTT, with more Standard deviation and max values than the previous tests.

Packet Loss	Mean (ms)	SD (ms)	Max (ms)	Min (ms)
0.5 %	11.7	4.3	904	11
1 %	11.9	7.6	2,960	11
5 %	15.6	44.1	2,882	11
10 %	33.3	103	4,750	8
20 %	269	568	8,874	7

Table 7.11: RTT Latency (ms), with different emulated packet loss values (10 minute runs).

The max value increases by a lot as the packet loss increases, as it has a snowball effect. If the message is lost multiple times (in consecutive retries), the max values will increase. With more packet loss, this is more probable, explaining why the max values are continuously increasing.

The following two tables show the tests for packet loss plus 10 ms RTT additional latency. Table 7.12 shows the number of messages consumed by the Kafka Consumer, and Table 7.13 shows the RTT latency values measured.

Packet Loss	Mean	SD	Max	Min
0.5 %	29,574	140	29,714	29,384
1 %	28,196	2,360	29,424	23,984
5 %	25,415	1,180	26,670	23,755
10 %	21,438	1,194	22,865	19,929
20 %	12,329	2,372	15,071	9,874

Table 7.12: Total number of messages consumed by the Kafka Consumer, with different emulated packet loss values and 10 ms additional RTT Latency, during 10 minute runs.

For packet loss values with 10 ms RTT latency, in terms of messages consumed, the platform performed well until 5% packet loss, with 10% and 20% packet loss scenarios getting six thousand messages less comparing to the scenarios with packet loss and no additional latency.

Packet Loss	Mean (ms)	SD (ms)	Max (ms)	Min (ms)
0.5 %	22	13	2,573	21
1 %	22.5	17.7	2,655	21
5 %	40	129	6,920	17
10 %	106	317	9,317	19
20 %	1,988	3,985	82,268	18

Table 7.13: RTT Latency (ms), with different emulated packet loss values and 10 ms additional RTT Latency (10 minute runs).

For RTT latency results, 0.5% to 1% scenarios performed well, but 5% and 10% packet loss scenarios started to have much higher latencies and standard deviations from the mean. 20% scenario took almost ten times more time to receive the messages than packet loss with no additional latency scenario, in both the mean and max values.

The small amount of added latency has severe consequences on the performance of the communication platform. This can be attributed to a configuration of the reliability policy

of the DDS publisher, how much time the publisher waits before retrying to send lost packets. If too small, the publisher will flood the channel with messages. If too big, the latency in lost packets increases.

7.4.4 Functional tests

Table 7.14 shows what functional requirements were implemented or not.

ID	Implemented?
RF1	Yes
RF2	Yes
RF3	Yes
RF4	Yes
RF5	Yes
RF6	Yes
RF7	Yes
RF8	Yes
RF9	No
RF10	No

Table 7.14: Functional requirements implemented

Both RF1 and RF2 are covered by the communication components created. BUBBLES' services interact via Kafka by means of a Kafka Consumer and a Kafka Publisher (Core broker in the test environment does both). UAS can interact via the communication agent to send messages to the Edge broker, which will forward its messages to the Core brokers where BUBBLES' services can consume the information.

RF3 is also implemented because the MAVLink allows the communication agent to receive information for all stages of the drone flight, including prior to take-off and after landing.

The architecture covers RF4. The decoupling between functions for edge and core brokers allows different providers to use their solutions while communicating to a shared system that might be centralized or not (Kafka organizations).

RF5 is covered by the protocols used, and both protocols allow point-to-multipoint communications.

RF6 was validated by listening to the packets via WireShark and seeing if they were encrypted. They were encrypted as both DDS:Security and Kafka encryption were configured.

RF7 was validated by trying to listen to the Kafka and DDS Topic by a not authenticated user. Authentication by Certificate Authorities was configured on both Kafka and DDS to support this functional requirement.

RF8 was validated by trying to listen to a Kafka and DDS topic without the proper authorization. This functional requirement was supported by configuring Access Control permission, in tandem with the certificates, on both Kafka and DDS.

RF9 and RF10 were not implemented.

This page is intentionally left blank.

Chapter 8

Conclusion

From the test results, it was concluded that functional requirements from 1 to 8 were implemented. For the non-functional requirements, the communication platform faired reasonably well in terms of throughput and latency.

Scalability values point to the consumption message throughput increasing as more Edge and Core brokers are added. In the worst-case scenario, the results show that with 20 drones and only two Edge and Core brokers, the U-Space services consume close to 650 thousand messages in ten minutes.

Latency tests with a controlled message rate on the ten-drone scenario using one topic measured approximately 2 ms RTT latency time. In the twenty-drone scenario, over its limit, the communication platform still has 56 ms RTT latency time, which is encouraging.

The most significant point for the communication system is the excellent latency achieved in UAS-Edge communications. With a mean of 0.2 ms and 0.3 ms, even in a twenty-drone scenario, with an incredibly small standard deviation.

One big surprise was the ability to maintain acceptable values for RTT latency when netEm removed packets in the order of 5% and 10% —achieving a mean of 15.6 ms and 33.3 ms RTT latency, respectively.

For UAV to Core communications, the RTT latency matches ICAO's 400 RCP, 240 RCP, and 120 RCP specifications. In some cases, like the RTT Latency test with ten drones or less, or the UAV-Edge test, the latency goes in line with the 60 RCP specifications, so it depends on the test scenario.

As for availability and continuity can be achieved by simply launching more edge brokers or core brokers. The architecture was designed to be scalable horizontally, and adding more brokers of any kind is not difficult.

This design covers the first two non-functional requirements (RNF1 and RNF2). From the UAS side, more edge brokers can be deployed without the need to inform the UAS. This is achieved by DDS broker-less architecture, which grants the ability to UAS to find new edge brokers on the fly.

On the side of U-Space services, more brokers can be added to guarantee replication between points of presence, and more consumers can be added to consumer groups to help consume the increasing number of drone messages.

The scalability tests also confirm the scalability of the communication platform. The platform was capable of handling more drones and more message throughput by just adding

Scenario	Delay Tracking Avrg (s)	Delay Tracking Std. dev (s)	Delay Separator Avrg (s)	Delay Separator Std. dev (s)
1	4	2	4	2
2	2	1	2	1
3	1	0.5	1	0.5

Table 8.1: Proposed performance values for scenarios (from [22]).

one of each broker.

In terms of integrity, both DDS and Kafka can offer at least once semantics. This guarantees the delivery of at least one message from one end to the other. In addition, the architecture was designed to accept distribution between points of presence, with no single point of failure. This further helps to guarantee the required integrity values.

The QoS level configurations, plus message replication, resulted that, in the tests, with a set of specific network configurations, the communication platform was able to guarantee ICAO's integrity levels defined by the RCP within the test time window (10min).

In a recent document published by the BUBBLES project [22], a new set of communication performance values were made available, as shown in Table 8.1.

The strictest value is for 1 second, and it is end-to-end contrary to the RTT values measured and from ICAO. This is easily achievable by the communication system, as only in really degraded conditions, like minimal bandwidth or extreme packet loss, the communication platform takes longer than 1 second on one-way measurements (RTT superior to 2000 ms).

In the end, eight functional requirements (RN1 to RN8) and at least five non-functional requirements (RNF1 to RNF5) were implemented. These requirements include all the MUST have requirements.

For the reasons stated above, in combination with completing everything within the deadline, the project can be considered a success, successfully offering a solution of TRL 3.

8.1 Future work

As for future work, the system has a lot to mature. The configuration for Kafka batch size needs more fine-tuning to find the balance between communication latency and bandwidth required for the traffic coming in and out of the Kafka Broker. The DDS retry wait time must also be finned-tune to stop drones from flooding the queues with retries in more dire situations.

Potential zero loss recovery is achievable with DDS automatic discovery. Having two edge brokers listening to the same UAS will ensure that contact with the drone is maintained even if one crashes. However, this brings problems of message duplication. DDS QoS properties or Kafka Idempotence requests, or both, can solve the problem, but it was not investigated during the thesis duration.

Ways to differentiate between geographical regions on the edge brokers should be implemented. DDS has virtual partitions that could be implemented to separate specific topics within the geographical section to better balance loads. As more clusters of Kafka brokers are added, replication between them should also be configured to guarantee even better data integrity.

Another group at the University of Coimbra, also working on the BUBBLES project, is developing a solution to inject other types of faults on the side of PX4, GPS telemetry corruption and/or manipulation. The test environment should also integrate with this tool that is currently being developed.

With the communication platform, BUBBLES will soon enter the first MEDUSA validation phase in the bigger scheme of things, with three planned exercises. This will help further validate the communication platform. The test environment created will also be used as a base for one of the exercises (with more realistic values like the ones concluded in the subsection Network Profiles).

After the validation exercises, new values for communication performance will be obtained, which will be used to create new requirements, and see the needed changes required to achieve those requirements. If changes are required, the process will be redone, doing the appropriate fixes to the architecture if needed, and develop new features to support any functional requirements that might arise from the evolution of the U-Space Services.

This Flow will be repeated (throughout MEDUSA's validation exercises) until the U-Space services, including the Separation Management service and the communication platform, are ready to be deployed on an industrial level.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015. ISSN 1553-877X. doi: 10.1109/COMST.2015.2444095.

[2] Dipa Soni and Ashwin Makwana. A SURVEY ON MQTT: A PROTOCOL OF INTERNET OF THINGS(IOT). April 2017.

[3] Andrew Banks, Rahul Gupta, Ed Briggs, and Ken Borgendale. MQTT Version 5.0. Technical report, OASIS Standard, March 2019.

[4] Vineet John and Xia Liu. A Survey of Distributed Message Broker Queues. April 2017.

[5] IBM Garage Event-Driven Reference Architecture – Kafka Overview. https://ibm-cloud-architecture.github.io/refarch-eda/technology/kafka-overview/.

[6] Data Distribution Service (DDS). *Object Management Group*, page 180, April 2015.

[7] Vlastimil Kriz and Petr Gabrlik. UranusLink - Communication Protocol for UAV with Small Overhead and Encryption Ability. *IFAC-PapersOnLine*, 48(4):474–479, January 2015. ISSN 2405-8963. doi: 10.1016/j.ifacol.2015.07.080.

[8] IMT-2020 Background.

[9] SITL Simulator (Software in the Loop) — Dev documentation. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[10] Simulation | PX4 User Guide. https://docs.px4.io/master/en/simulation/.

[11] Velos-UAV_Camera_4-1024x455.jpg (JPEG Image, 1024 × 455 pixels). http://velosuav.com/wp-content/uploads/2015/11/Velos-UAV_Camera_4-1024x455.jpg.

[12] Betsy Lillian. Skyfront Boasts Multi-Rotor UAV Flight of More Than 4.5 Hours. https://unmanned-aerial.com/skyfront-boasts-multi-rotor-uav-flight-4-5-hours, September 2017.

[13] BAAM.Tech Futura VTOL - Design with BAAM.Tech, May 2021.

[14] Condé Nast. Amazon's UK drone delivery tests will start 'right away'. *Wired UK*. ISSN 1357-0978.

[15] Apache Kafka. https://kafka.apache.org/protocol.html, .

[16] Pinchen Cui. Comparison of IoT application layer protocols., 2017.

[17] The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification, June 2014.

[18] Serialization · MAVLink Developer Guide. https://mavlink.io/en/guide/serialization.html.

[19] Cherub Dim, Francis Nabor, Giancarlo Santos, Martin Schoeler, and Alvin Chua. Novel Experiment Design for Unmanned Aerial Vehicle Controller Performance Testing. *IOP Conference Series: Materials Science and Engineering*, 533:012026, May 2019. ISSN 1757-899X. doi: 10.1088/1757-899X/533/1/012026.

[20] Zakariya Laftit and Diana Mardare. SESAR 2020 - 763601 - D2.4 Overall System architecture – Update to D2.2. page 66, 2018.

[21] Manual on Required Communication Performance (RCP). page 42.

[22] Deliverable D6.5 (to be published), April 2021.

[23] Drone Types: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL | AUAV. https://www.auav.com.au/articles/drone-types/, .

[24] Communication-from-the-commission-to-the-european-parliament-and-the-council.

[25] EU Business in Japan |. https://www.eubusinessinjapan.eu/.

[26] SESAR Joint Undertaking | European Drones Outlook Study. https://www.sesarju.eu/node/2951, .

[27] Description of work document (DoW), October 2019.

[28] SESAR JU . U-space research innovation results.pdf, 2020.

[29] SESAR Joint Undertaking | U-space. https://www.sesarju.eu/U-space, .

[30] Defining the BUilding Basic BLocks for a U-Space SEparation Management Service | BUBBLES Project | H2020 | CORDIS | European Commission. https://cordis.europa.eu/project/id/893206.

[31] Advanced Message Queuing Protocol. *IEEE Internet Computing*, 10(6):87–89, November 2008. ISSN 1089-7801. doi: 10.1109/MIC.2006.116.

[32] Which protocols does RabbitMQ support? — RabbitMQ. https://www.rabbitmq.com/protocols.html.

[33] Rabbitmq/rabbitmq-amqp1.0. RabbitMQ, April 2021.

[34] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, October 2012.

[35] IBM Docs. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/ibm-mq/9.2?topic=reliability-message-delivery-amqp, June 2021.

[36] Apache Kafka. https://kafka.apache.org/documentation.html#design, .

[37] Kafka Incubation Status - Apache Incubator. https://incubator.apache.org/projects/kafka.html, .

[38] Kafka Needs No Keeper - Removing ZooKeeper Dependency. https://www.confluent.io/blog/removing-zookeeper-dependency-in-kafka, .

[39] Stipe Celar, Eugen Mudnic, and Zeljko Seremet. State-Of-The-Art of Messaging for Distributed Computing Systems. pages 0298–0307. January 2016. ISBN 978-3-902734-08-2. doi: 10.2507/27th.daaam.proceedings.044.

[40] Philippe Dobbelaere and Kyumars Sheykh Esmaili. Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. pages 227–238, June 2017. doi: 10.1145/3093742.3093908.

[41] Sanika Raje. Performance Comparison of Message Queue Methods. doi: 10.34917/16076287.

[42] Ian Noel McAteer, Muhammad Imran Malik, Zubair Baig, and Peter Hannay. Security vulnerabilities and cyber threat analysis of the AMQP protocol for the internet of things. 2017. doi: 10.4225/75/5A84F4A695B4C.

[43] Heiko Koziolek, Sten Grüner, and Julius Rückert. A Comparison of MQTT Brokers for Distributed IoT Edge Computing. In Anton Jansen, Ivano Malavolta, Henry Muccini, Ipek Ozkaya, and Olaf Zimmermann, editors, *Software Architecture*, volume 12292, pages 352–368. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58922-6 978-3-030-58923-3. doi: 10.1007/978-3-030-58923-3_23.

[44] Joseph Anthraper and Jaidip Kotak. Security, Privacy and Forensic Concern of MQTT Protocol. *SSRN Electronic Journal*, January 2019. doi: 10.2139/ssrn.3355193.

[45] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A Distributed Messaging System for Log Processing. page 7.

[46] Apache Kafka. https://kafka.apache.org/cve-list, .

[47] Florian Pinzel, Jörg Holfeld, Andrej Olunczek, Paul Balzer, and Oliver Michler. V2V- and V2X-Communication data within a distributed computing platform for adaptive radio channel modelling. In *2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 1–6, June 2019. doi: 10.1109/MTITS.2019.8883347.

[48] Roshan Sedar, Francisco Vázquez-Gallego, Ramon Casellas, Ricard Vilalta, Raul Muñoz, Rodrigo Silva, Laurent Dizambourg, Antonio Eduardo Fernández Barciela, Xavier Vilajosana, Soumya Kanti Datta, Jérôme Härri, and Jesus Alonso-Zarate. Standards-Compliant Multi-Protocol On-Board Unit for the Evaluation of Connected and Automated Mobility Services in Multi-Vendor Environments. *Sensors*, 21(6): 2090, January 2021. doi: 10.3390/s21062090.

[49] Yakusheva Nadezda. A Safe Intelligent Driver Assistance System in V2X Communication Environments based on IoT. page 140.

[50] Soumya Kanti Datta, Jean-Luc Dugelay, and Christian Bonnet. IoT Based UAV Platform for Emergency Services. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 144–147, October 2018. doi: 10.1109/ICTC.2018.8539671.

[51] Kamil Nuther Saib. *A Distributed Architecture for Unmanned Aerial Systems Based on Publish/Subscribe Messaging and Simultaneous Localisation and Mapping (SLAM) Testbed*. Thesis, 2017.

[52] Amartya Mukherjee, Nilanjan Dey, and Debashis De. EdgeDrone: QoS aware MQTT middleware for mobile edge computing in opportunistic Internet of Drone Things. *Computer Communications*, 152:93–108, February 2020. ISSN 0140-3664. doi: 10.1016/j.comcom.2020.01.039.

[53] RTI Connext DDS - What is Discovery? https://community.rti.com/static/documentation/connext-dds/6.0.0/doc/manuals/connext_dds/html_files/RTI_ConnextDDS_CoreLibraries_UsersManual

[54] DDS Security Version 1.0, September 2016.

[55] Mavlink-router/mavlink-router. mavlink-router, June 2021.

[56] UAVCAN Development Team. Uncomplicated Application-layer Vehicular Computing And Networking. https://uavcan.org/.

[57] UAVCAN_Specification_v1.0-beta, January 2021.

[58] Matrix.org. https://matrix.org, .

[59] Matrix Specification. https://matrix.org/docs/spec/, .

[60] Stefan Profanter, Ayhun Tekat, Kirill Dorofeev, Markus Rickert, and Alois Knoll. OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. February 2019. doi: 10.1109/ICIT.2019.8755050.

[61] Ivan Vidal, Paolo Bellavista, Victor Sanchez-Aguero, Jaime Garcia-Reinoso, Francisco Valera, Borja Nogales, and Arturo Azcorra. Enabling Multi-Mission Interoperable UAS Using Data-Centric Communications. *Sensors (Basel, Switzerland)*, 18 (10), October 2018. ISSN 1424-8220. doi: 10.3390/s18103421.

[62] Thomas White, Michael N. Johnstone, and Matthew Peacock. An investigation into some security issues in the DDS messaging protocol. 2017. doi: 10.4225/75/5A84FCFF95B52.

[63] Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgui. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access*, 7:87658–87680, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2924410.

[64] Navid Khan, Noor Zaman, Sarfraz Brohi, and Anand Nayyar. Emerging use of UAV's: Secure communication protocol issues and challenges. pages 37–55. January 2020. ISBN 978-0-12-819972-5. doi: 10.1016/B978-0-12-819972-5.00003-3.

[65] Karel Domin, Eduard Marin, and Iraklis Symeonidis. Security Analysis of the Drone Communication Protocol: Fuzzing the MAVLink protocol. page 7.

[66] IEEE 802.11, The Working Group Setting the Standards for Wireless LANs. https://www.ieee802.org/11/, .

[67] IEEE 802.11, The Working Group Setting the Standards for Wireless LANs. https://www.ieee802.org/11/, .

[68] Mauro Callejas Cuervo, Manuel Vélez-Guerrero, and Andrea Aldana. Characterization of Wireless Data Transmission over Wi-Fi in a Biomechanical Information Processing System. *Revista Facultad de Ingeniería*, 29:e10228, November 2019. doi: 10.19053/01211129.v29.n54.2020.10228.

[69] Junxian Huang, Feng Qian, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. *MobiSys'12 - Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, June 2012. doi: 10.1145/2307636.2307658.

[70] Performance of Wireless Networks: Mobile Networks - High Performance Browser Networking (O'Reilly). https://hpbn.co/mobile-networks/.

[71] Najett Neji and Tumader Mostfa. Communication technology for Unmanned Aerial Vehicles: A qualitative assessment and application to Precision Agriculture. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 848–855, Atlanta, GA, USA, June 2019. IEEE. ISBN 978-1-72810-333-4. doi: 10.1109/ICUAS.2019.8797879.

[72] Software framework for long-distance transportation drones. https://research.tue.nl/en/studentTheses/software-framework-for-long-distance-transportation-drones.

[73] Marco Giordani and Michele Zorzi. Non-Terrestrial Networks in the 6G Era: Challenges and Opportunities. *IEEE Network*, 35(2):244–251, March 2021. ISSN 0890-8044, 1558-156X. doi: 10.1109/MNET.011.2000493. Comment: The paper has been accepted for publication in IEEE Network Magazine. 8 pages, 4 figures, 2 tables.

[74] Satoshi Utsumi, Salahuddin Muhammad Salim Zabir, and Sumet Prabhavat. A new explicit congestion notification scheme for satellite IP networks. *Journal of Network and Computer Applications*, 75:169–180, November 2016. ISSN 1084-8045. doi: 10.1016/j.jnca.2016.08.027.

[75] Thomas Gräupl. LDACS A/G Specification. page 211, 2019.

[76] Thomas Gräupl, Max Ehammer, and Carl-Herbert Rokitansky. L-DACS 1 data link layer design and performance. pages 1–12, June 2009. doi: 10.1109/ICNSURV.2009.5172855.

[77] L-band Digital Aeronautical Communications System (LDACS). https://tools.ietf.org/id/draft-maeurer-raw-ldacs-01.html#name-ldacs-data-rates.

[78] RECOMMENDATION ITU-R M.2012-4 - Detailed specifications of the terrestrial radio interfaces of International Mobile Telecommunications-Advanced (IMT-Advanced). page 237.

[79] ITU: Committed to connecting the world. https://www.itu.int:443/en/Pages/default.aspx.

[80] A. Abioye, M. Joseph, and Hendrik Ferreira. Comparative Study of 3G and 4GLTE Network. *Journal of Advances in Computer Networks*, 3:247–250, January 2015. doi: 10.7763/JACN.2015.V3.176.

[81] M Fricke, A Heckwolf, Ralf Herber, Ralf Nitsch, Silvia Schwarze, Stefan Voss, and Stefan Wevering. Requirements of 4G-based mobile broadband on future transport networks. *Journal of Telecommunications and Information Technology*, 2/2012:19–26, January 2012.

[82] M. Solera, M. Toril, I. Palomo, G. Gomez, and J. Poncela. A Testbed for Evaluating Video Streaming Services in LTE. *Wireless Personal Communications*, 98(3):2753–2773, February 2018. ISSN 0929-6212, 1572-834X. doi: 10.1007/s11277-017-4999-0.

[83] Cise Midoglu, Konstantinos Kousias, Özgü Alay, Andra Lutu, Antonios Argyriou, Michael Riegler, and Carsten Griwodz. Large scale "speedtest" experimentation in Mobile Broadband Networks. *Computer Networks*, 184:107629, January 2021. ISSN 1389-1286. doi: 10.1016/j.comnet.2020.107629.

[84] Pedro Casas and Raimund Schatz. Quality of Experience in Cloud services: Survey and measurements. *Computer Networks*, 68:149–165, August 2014. ISSN 1389-1286. doi: 10.1016/j.comnet.2014.01.008.

[85] G. Patounas, X. Foukas, A. Elmokashfi, and M. K. Marina. Characterization and Identification of Cloudified Mobile Network Performance Bottlenecks. *arXiv:2007.11472 [cs]*, July 2020. Comment: 17 pages, 16 figures, documentclass[journal,comsoc]{IEEEtran}, corrected title.

[86] Pablo Oliver Balsalobre, Matias Toril, Salvador Luna-Ramírez, and Rafael Garaluz. A system testbed for modeling encrypted video-streaming service performance indicators based on TCP/IP metrics. *EURASIP Journal on Wireless Communications and Networking*, 2017, December 2017. doi: 10.1186/s13638-017-0999-8.

[87] Release 12. https://www.3gpp.org/specifications/releases/68-release-12, .

[88] Release 13. https://www.3gpp.org/release-13, .

[89] The use of the terrestrial component ofInternational Mobile Telecommunicationsfor narrowband and broadband machinetypecommunications. page 11.

[90] Minimum requirements related to technical performance for IMT-2020 radio interface(s). *International Telecommunication Union*, page 11.

[91] Peter Rost, Christian Mannweiler, Diomidis S. Michalopoulos, Cinzia Sartori, Vincenzo Sciancalepore, Nishanth Sastry, Oliver Holland, Shreya Tayade, Bin Han, Dario Bega, Danish Aziz, and Hajo Bakker. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications Magazine*, 55(5):72–79, May 2017. ISSN 1558-1896. doi: 10.1109/MCOM.2017.1600920.

[92] Summary of Rel-16 Work Items (Release 16). Technical report, 3rd Generation Partnership Project.

[93] Millimeter Wave - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/engineering/millimeter-wave.

[94] Guntis Ancāns, A. Stafecka, V. Bobrovs, Arnis Ancans, and Jelena Caiko. Analysis of Characteristics and Requirements for 5G Mobile Communication Systems. *Latvian Journal of Physics and Technical Sciences*, 54, August 2017. doi: 10.1515/lpts-2017-0028.

[95] Pingzhi Fan, Jing Zhao, and Chih-Lin I. 5G high mobility wireless communications: Challenges and solutions. *China Communications*, 13(Supplement2):1–13, 2016. ISSN 1673-5447. doi: 10.1109/CC.2016.7833456.

[96] Cristina Gabriela Gheorghe, Dan Alexandru Stoichescu, and Radu Dragomir. Latency requirement for 5G mobile communications. In *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–4, June 2018. doi: 10.1109/ECAI.2018.8679058.

[97] ArduPilot :: About. https://ardupilot.org/about.

[98] The Dronecode Foundation - We are setting the standards in the drone industry with open-source - Join the Community! https://www.dronecode.org/, .

[99] U-space Concept of Operations vol2, 25/ 10/2019.

[100] JARUS guidelines on Specific Operations Risk Assessment (SORA), January 2019.

[101] CORUS team. CORUS Intermediate U Space ConOps Vol3 Annex D MEDUSA.

[102] Andrew Craddock and DSDM Consortium. MOSCOW PRIORITISATION. In *The DSDM Agile Project Framework*. 2014. ISBN 978-0-9544832-9-6.

[103] CORUS ConOps vol2, September 2019.

[104] eProsima: RTPS/DDS Experts. https://www.eprosima.com/.

[105] NetEm - Network Emulator at Linux.org. https://www.linux.org/docs/man8/tc-netem.html.

[106] iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. https://iperf.fr/.

Appendices

This page is intentionally left blank.

Appendix A - Drones composition

Depending on the different objective types of drones and components can be chosen to deliver better the task at hand. There are plenty of different models and a few different types of UAVs to pick from nowadays.

One of the most known types of drones is the Multi Rotor, with the Quadcopter belonging to this category, that is one that most people think of when they think of drones. The idea of designing rotorcrafts started very similarly to traditional helicopters. As time went by, UAVs would become smaller and lighter, aiming for a low-weight aerodynamic structure. With that goal in mind, the idea of multi-rotor drones evolved rapidly.

A list of the four major types of drones will be presented, describing how they work and the different mechanisms that they use, and how they propel themselves during take-off and flight.

Single Rotor

Single rotor drones are the most basic type of drone. As the name implies, they only have one propeller (i.e., rotor). Although they have only one rotor, they can sometimes deliver more thrust than their multi-rotor counterparts. In terms of power consumption, they have better efficiency than multi-rotor ones, and for even bigger endurance can be powered by a gas-based motor.

The reason that they are more efficient than Multi Rotor ones, there is an aerodynamics rule that states the larger the rotor blade's diameter and the slower it spins, the more efficient it is.

Figure 1: A Single Rotor drone (from [11]).

In addition to the oversized rotor, as it happens with the traditional helicopters, they have a smaller one on the tail to control the heading of the vehicle. Due to greater efficiency and more considerable thrust, they can carry heavier payloads. These types of drones can also hover thanks to the Vertical Take-Off and Landing (VTOL) capabilities the rotor provides. One downside is that they pose a more significant threat since it carries a bigger blade. Other downsides include the complexity of maneuvering the drone and its costs.

Multi-Rotor

Multi-rotor drones, as the single rotor ones, have VTOL capabilities. This is the main advantage of rotorcrafts.

They are the cheapest options among drone types and are the more accessible option for

people entering the field. As a result of the combination of multiple rotors, it offers greater control over its position facilitating new users that do not have a lot of skills.

Figure 2: A Multi-Rotor drone (from [12]).

Although the most common multi-rotor is the Quadcopter, they can have a different number of rotors. Various models can be found with three (tricopter), four (Quadcopter), six (hexacopter), and eight (octocopter) rotors. Following the aerodynamics rules stated earlier, the more rotors the drone has, the less efficient it is. In combination with their low cost and easy control mechanism, the abundance of these models invited many to use this kind of UAVs.

They are extremely good at small distances flights and have incredibly high maneuverability. This, combined with the capability of hovering locations, makes them suitable for photography or filming framing. One of the most significant downsides of these types of UAVs is their limited endurance and speed. They are not fit for long flights.

Fixed Wing

These drones have, as the name says, a fixed-wing style flight as opposed to the rotary-wing flight style of single or multi-rotor drones. This is most similar to what airplanes are nowadays.

Instead of propellers, they have wings that allow for a vertical lift, and this means that they only need a horizontal force to keep moving forward.

Figure 3: A Fixed Wing drone (from [13]).

These fixed-wing drones are able to stay airborne for a lot longer, reaching almost 16 hours of continuous flying. Since these types of drones are more compact and resistant, they can

also withstand more tough environmental conditions. However, they do not offer as good maneuverability as single or multi router drones, but the long endurance makes them ideal for long-range flights.

One of the most significant downsides is the horizontal take-off technique; to take off and to land, they need a runaway track. In addition to needing a track, the landing may also damage the drone, which can bring additional costs to the maintenance of the UAV.

Fixed Wing Hybrid

These types of drones try to merge the best of fixed-wing flight: the low consumption of energy for flights with one of the best things of rotorcrafts that is the ability to VTOL and hover locations. Some of them have a rotor strategically added to an already fixed wind model; some other models tilt the wings for lift-off and landing, with the wings staying folded on the ground and aligning during flight.

Figure 4: A Fixed Wing Hybrid drone (from [14]).

They were theorized and experienced with, but all ended up in failures because they proved too difficult to fly and be controlled by human interaction. With the growth of autopilots, the software can deal with the more complex calculations facilitating the user's job.

These types of UAVs are designed to incorporate the advantages of both VTOL and the fast-flying Fixed-Wing. However, this type of UAV is complex to create, making them a bit more expensive than the former types.

Components

Currently, UAVs come packed with a high degree of technological equipment that helps broaden the actual functionalities of drones. Most drones that are used come equipped with a variety of electronic devices that produce a lot of data about the current status of the air vehicle. What follows is a list of the most common components in most if not in all, drones today.

Magnetometers

They are the compasses of aviation. It can measure both the strength and direction of magnetic fields. By measuring the Earth's magnetic field, it can extract the object's

	Pros	Cons	Typical Users
Single-Rotor	-VTOL and hover flight. -Long endurance (with gas power). -Heavier payload.	-More dangerous. -Harder to fly, more training required. -Expensive.	Aerial LIDAR laser scanning.
Multi-Rotor	-Accessibility. -Ease of use. -VTOL and hover flight. -Can operate in a confined area.	-Short flight times. -Small payload capacity.	Aerial Photography and Video Aerial Inspection.
Fixed-Wing	-Long endurance. -Larger coverage area. -Fast flight speed.	-Launch and recovery needs a lot of space. -No VTOL/hover. -Harder to fly, more training required. -Expensive.	Aerial Mapping, Pipeline and Power line inspection.
Fixed-Wing Hybrid	-VTOL and long-endurance flight.	-Not perfect at either hovering or forward flight. -Most recent to appear.	Drone delivery.

Table 2: Summary of the differences between types of drones (from [23]).

orientation by applying a mathematical formula. They are prone to error, and it receives some external noise from the other devices on board, so they are better used in combination with a Global Navigation Satellite System (GNSS), like GPS, or an Inertial Measurement Unit (IMU).

They are critical in rotor-based drones because they can provide orientations when hovering or in situations where the GNSS cannot provide accurate information. For fixed-wing, it is not so important. Due to the lack of maneuverability, the GNSS can provide somewhat better information. But some mathematical errors may arise when applying the formula by not considering the yaw angle that drones may experience. Or the signal of the GNSS may lack. For those reasons, Magnetometers are also used in fixed-wing.

GPS

The GPS module offers the positioning of the drone in the 3d space. Often this module combines the information received from the GNSS and magnetometer. It is usually a bit distanced from the other components to minimize interference from the other onboard components.

It measures the drone's location by calculating how long a signal takes to travel from a satellite. However, GPS modules are rather inaccurate and will only give you a position within 5m for the values of latitude and longitude.

It can also provide the elevation of the drone. Albeit that elevation measuring from GNSS alone does not have good accuracy, data being accurate in the range of 5 to 10 meters post corrections. This GPS module is what allows some drones to register a failsafe home

point. In case of emergency, the UAV can fly back to a safe location. This module is an essential requirement for waypoint navigation which is the base for autonomous flight modes. Without it, a concept like U-Space would never be possible.

Barometer

This device is capable of measuring atmospheric pressure. The number of atmospheres (unit of measurement for atmospheric pressure) drops as the altitude increases; as altitude decreases, the number of atmospheres increases. The barometer is adjusted to read the changes in pressure to give accurate altitude readings.

These pressure sensors are so sensitive that they can detect the change in air pressure when your drone moves a few centimeters.

Pilot tube

A pilot tube is used to estimate the velocity of air. It is mainly used in fixed-wing models to help coordinate the wings. The wings must be positioned a certain way depending on the airspeed passing the vehicle to keep moving.

It can determine how fast the air is passing through the aircraft by comparing the dynamic (kinetic) and static pressures via the pilot tube. This is extremely important because these types of drones use this same airflow to generate lift (if the drone is flying too slowly, the plane could end stalling and crashing).

IMU

This device is essential in UAVs for control, stabilization, guidance, and correction. The inertial measurement unit is an electronic device that can measure the orientation, angular rate, and specific force of a body using accelerometers and gyroscopes to measure acceleration and rotation.

Some IMU also includes magnetometers to assist with calibration against orientation drift. It provides essential navigation information to the central flight control system.

Distance sensors

These types of sensors can be ultrasonic distance sensors (e.g., SONAR) or light-based distance sensors. Either by lasers or by sending a sound and measuring how long it takes to bounce back, most sensors are used aiming down so they can measure their distance to the relative ground (the closest object directly down).

By keeping tabs on the current relative altitude, drones are able to change their elevation in case of flying below a minimum safe distance to the ground. This is especially important in populated areas where the relative ground will keep changing, so as the terrain will keep changing height so will the UAV.

Flight controller

The flight controller is considered the brain of the whole operation. This is where all information ends up so it can be used to determine the current status of the drone. It reads all the sensor data and calculates the best command to send to the actuators. The actuators can be motors powering the flight, cameras, speakers, or other payloads.

It maintains the drone flying and also controls the autonomous functions of the drone. In the processor of the flight, the controller resides the autopilot. The autopilot is a system that does more than what the name entails. Although the autopilot is indeed a system that makes the drone fly autonomously to waypoints, it is not all. It facilitates the skills necessary to fly the drone so that less skilled users are able to fly these complex machines.

Because flying requires attention to a whole set of environmental readings, the autopilot will gather all the information necessary with the help of a variety of sensors. The processor crunches the numbers and makes the computations needed to transform all the data available into the required instructions to keep the drone flying.

The autopilot plays a vital role in the flight of these systems as it coordinates the air vehicle with the instructions sent from either human input or a control station. Allowing for better control without all the precise and challenging instructions that are needed to fly the drone. With this, flying can be achieved with minimal to no-human input.

Appendix B - BUBBLES services

Figure 5 shows BUBBLES' architecture. To help the reader better understand how the BUBBLES services work, an explanation directly extracted from the project DoW [27] is next reproduced.

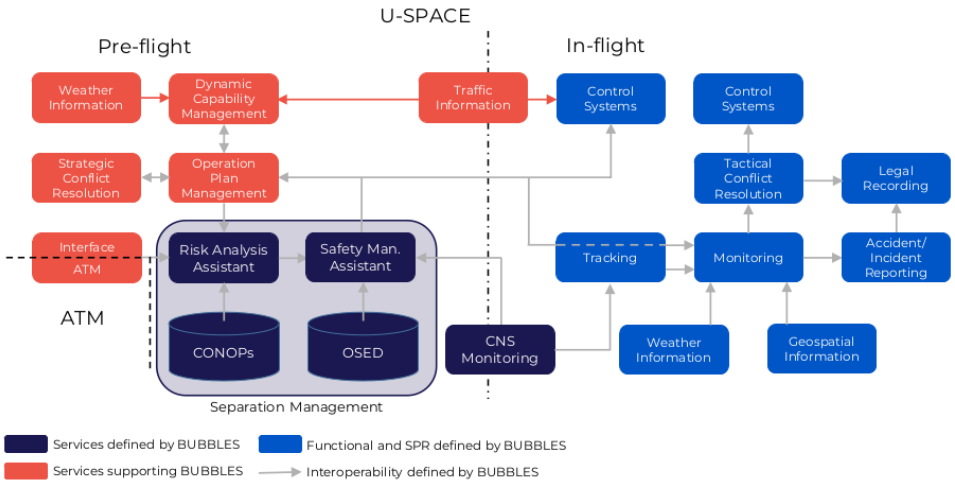

Figure 5: BUBBLES's concept.

The BUBBLES U-Space Separation Management service will be composed of four core technical blocks and one standardization block. The technical blocks are:

The generic CONOPs Catalogue. This Catalogue will contain a set of CONOPs describing from the users' perspective how the UAS operations will occur in a given Operational Environment, with risk levels linked to them following standard procedures like the Specific Operational Risk Assessment (SORA).

The generic OSED Catalogue. This Catalogue will contain the generic OSED defined by the project. Each OSED will consist of a risk level (or a range of them), the required U-Space services, and the separation minima and methods associated with it. The separation minima and methods will be assigned based on a safety assessment using the MEDUSA proposed by the project CORUS and the collision risk analysis algorithms developed by the BUBBLES project.

The Risk Analysis Assistant. This block will correlate a given Operational Plan with the CONOPs stored in the generic CONOPs Catalogue assigning to it the risk level of the most similar one, using algorithms developed by the BUBBLES project.

The Safety Management Assistant. This block will receive the risk level assigned to the Operation Plan by the Risk Analysis Assistant and will assign an OSED, taking into account the performance of the CNS systems and the separation minima and methods assigned using algorithms developed by the project.

Moreover, the U-Space Separation Management service will require an additional enabling block: the CNS Monitoring tools defined by BUBBLES to monitor the actual performance of the CNS systems in the VLL volume where a particular Operation Plan will take place.

Lastly, the implementation of the U-Space Separation Management service will require updating several already defined U-Space services and will involve others as they have been described (just updating their interfaces).

In addition to the technical blocks described in the paragraphs above, BUBBLES will also have a standardization block. This block will consist of drafts of: Required Performance Specifications for CNS systems. These specifications will define metrics to describe the CNS systems performance, the required thresholds and methods to assess the compliance with those thresholds. Guidelines to design the algorithms used to generate the separation instructions and SPR for the systems implementing them (covering the definition of metrics, thresholds, and compliance assessment methods). Requirements (including assessment methods) to guarantee that the information provided by the Geospatial and Weather Information services do not compromise safety.

Functionally, the BUBBLES U-Space Separation Management service will carry out the following sequence of operations (in bold italics we highlight those U-Space services that are going to be used as defined by CORUS with updated interfaces while normal bolds characters are used to denote those components or U-Space services that are going to be defined or updated by the project;):

In the pre-flight phase: 1. The Operation Plan Management service will pass to the Risk Analysis Assistant the Operation Plan description previously introduced by the UAS operator, including the operation area co-ordinates.

2. The Risk Analysis Assistant will correlate the Operation Plan description received with those stored in the generic CONOPs Catalogue and will assign a risk level to it.

3. The Safety Management Assistant will receive the operation area co-ordinates along with the assigned risk level from the Risk Analysis Assistant and, taking into account the available CNS systems in the operation area and their performance (provided by the CNS Monitoring tools), will assign one generic OSED out of those stored in the generic OSED Catalogue. The assigned OSED, including the applicable separation minima and methods, will be sent to the Operation Plan Management service.

The Operation Plan Management service, depending on the separation method associated to the received OSED, will send the updated Operation Plan:

a) to the Strategic Conflict Resolution service (in case of 'procedural' separation) and, after any possible conflict (including with manned aircrafts in Zu 5 volumes) has been resolved, to the UAS Control system.

b) to the ATM systems through the Interface to ATM, for UAS flying in Za 5 volumes.

c) to the UAS DAA system (in case of 'tactical self-separated' separation); the Remain well Clear (RwC) function will be provided by the DAA capability using the situational awareness obtained from the Traffic Information service or generated on-board, or d) to the U-Space Monitoring service (in case of 'tactical ground-based' separation).

In the In-flight phase: 1. In case of 'tactical ground-based' separation, the Monitoring service will receive the tracks obtained by the Tracking service (including the UAS/manned aircrafts), as well as the data provided Geospatial and Weather Information services and will check that all the UAS maintain the assigned separation minima. If it detects a potential infringement of the separation minima, it will issue an alarm to the Tactical Conflict Resolution service and will monitor whether the conflict is resolved.

2. The Tactical Conflict Resolution service will issue separation instructions to resolve any potential conflict detected, while preserving the overall safety level. The separation instructions will be timestamped and sent to the Legal Recording service, as well as the eID of the involved UAS/manned aircraft.

3. In case of 'tactical self-separation', the UAS DAA system will check that the UAS maintains the assigned separation minima based on the situational awareness obtained from internal or external sources. If it detects a potential infringement of the separation minima, it will issue separation instructions to the UAS Control system.

4. In both cases (ground-based and self-separated), if the Monitoring service detects that the applied separation instructions are not effective, a timestamped report with the eID of the involved UAS/manned aircraft will be sent to the Accident/Incident Reporting service.

5. Also, in both cases (ground-based and self-separated), the CNS Monitoring tools will monitor the CNS systems performance and will issue warnings to the Safety Management Assistant when the values of the performance metrics change. Upon the received performance information, it will assess whether the separation minima and/or methods need to be updated in order to preserve the overall safety level. If so, the updated values will be sent to the UAS DAA system or to the Monitoring service through the Operation Plan Management service.

6. In case that a performance degradation is detected in the surveillance systems, the values of the performance metrics will be fed into the Tracking service so that the tracking algorithms can be adapted to them.

7. Optionally, the separation minima assigned by BUBBLES may be used by the Dynamic Capability Management service to dynamically optimize the use of the VLL.