Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

2024

NATHAN
CAMPOS

MAKING THE VIRTUAL A REALITY: A FRAMEWORK TO CONNECT PHYSICAL DEVICES TO VIRTUAL WORLDS

Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

2024

NATHAN
CAMPOS

MAKING THE VIRTUAL A REALITY: A FRAMEWORK TO CONNECT PHYSICAL DEVICES TO VIRTUAL WORLDS

Dissertação apresentada ao IADE - Faculdade de Design, Tecnologia e Comunicação da Universidade Europeia, para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Computação Criativa e Inteligência Artificial, realizada sob a orientação científica do Professor Doutor João Alfredo Fazendeiro Fernandes Dias, *Professor Associado* da Universidade Europeia.

agradecimentos

Começo por agradecer a todos que acreditaram que um dia seria capaz de escrever uma tese, concluir um mestrado e de estar a contribuir para a comunidade académica. Vocês foram muitos ao longo de tantos anos, nunca acreditei em nenhum de vocês, mas aparentemente estavam certos o tempo todo, talvez devo passar a vos ouvir mais.

Gostaria também de agradecer especialmente ao meu orientador, João Dias, que ao longo de todo esse processo me ajudou a manter o meu foco e me auxiliou e guiou nos momentos em que mais precisei, além de, até o último momento, foi capaz de conter suas emoções e suportar meus testes de paciência e entregas de trabalho feitas sempre a última da hora.

Um agradecimento especial a minha esposa, Luiza, que esteve junto de mim durante todo esse processo e foi a maior ajuda emocional que já tive em minha vida para a conclusão de um projeto. Muito obrigado por todos estes anos e por sempre ter estado ao meu lado, me auxiliando em minhas decisões e me ajudando a cada passo ao longo de nossas vidas juntos.

Quero estender um agradecimento especial também aos meus pais, Ladislau e Maria, por sempre terem acreditado em mim e terem me auxiliado em cada passo ao longo do caminho para poder chegar até onde atualmente me encontro. Nada disso seria possível sem vocês.

Não posso deixar de agradecer a todos os meus amigos que me suportaram ao longo destes anos, tanto no Brasil como em Portugal, também não seria possível eu estar aqui sem a ajuda de todos vocês.

Por fim gostaria de agradecer a imensa ajuda do meu gato, Xanino, que sempre esteve presente em minha vida desde 2019 e continua a ser uma companhia indispensável para todos os momentos. Sem suas brincadeiras e seu apoio emocional tudo seria muito mais difícil.

palavras-chave Aplicações Criativas; Internet das Coisas; Realidade Estendida; Framework de Software; Gêmeo Digital

resumo Recentemente, muitos avanços foram feitos nas áreas das tecnologias de Realidade Estendida (XR) e em aplicações criativas que aproveitam dos novos níveis de imersão e de *blending* de ambientes proporcionados por estas novas tecnologias. Muitas destas aplicações ainda estão ligadas a dispositivos de entrada tradicionais, como um rato, um ecrã tátil ou um comando, e também a dispositivos de saída muito simples, como ecrãs ou projetores. Com a democratização de soluções acessíveis da Internet das Coisas (IoT), existe uma oportunidade para integrar estes dispositivos na próxima geração de aplicações criativas, embora tais integrações possam ser dispendiosas ou envolver muito trabalho customizado para o seu desenvolvimento. Esta proposta visa colmatar esta lacuna e criar uma *framework* de software que possa ser utilizada pelos programadores de aplicações criativas para a integração de dispositivos IoT em seus projetos para a criação de gémeos digitais ou como dispositivos de entrada e saída para as suas aplicações, abrindo assim o desenvolvimento destas aplicações a todo um novo mundo de possibilidades. Ao longo deste trabalho de investigação, o protocolo de comunicação IoT MQTT, um padrão da indústria, foi testado e provou a sua viabilidade no fornecimento de meios fiáveis de comunicação de baixa latência para dispositivos de entrada sem fios, bem como a proposta pormenorizada de um esquema de emparelhamento para dispositivos IoT que lhes permite serem emparelhados com confiança com clientes automaticamente.

keywords Creative Applications; Internet of Things; Extended Reality; Software Framework; Digital Twin

abstract In recent times there were many advancements in Extended Reality (XR) technologies and creative applications that take advantage of the new levels of immersion and environment blending brought on by these new technologies. Many of these applications are still tied to traditional input devices, such as a mouse, touchscreen, or game controller, and also output to very mundane output devices such as screens or projectors. With the democratization of affordable Internet of Things (IoT) solutions, there is an opportunity for integrating these devices into the next generation of creative applications, although such integrations may be costly or involve a lot of custom work to develop. This proposal aims to bridge this gap and create a software framework that can be used by developers of such applications to leverage IoT devices for the creation of digital twins or as input and output devices for their applications, opening up software development to a whole new world of possibilities. Throughout this research work, the industry standard IoT communication protocol MQTT was tested and proved its viability in providing reliable means of low-latency communication for wireless input devices, as well as the detailed proposal of a pairing scheme for IoT devices that allows them to be matched with confidence to clients automatically.

Contents

Acronyms

AI Artificial Intelligence. 20

API Application Programming Interface. vii, 10–12, 27, 29, 30, 38

AR Augmented Reality. vii, 1, 4, 5, 7, 8

CAVE Cave Automatic Virtual Environment. 5

CoAP Constrained Application Protocol. vii, 12, 13, 51

DHCP Dynamic Host Configuration Protocol. 9, 29

DMX512 Digital Multiplex Protocol. 22

DoS Denial of Service. 17, 29

FIFO First in first out. 16

GPS Global Positioning System. 23

HMD Head Mounted Display. 5

HTTP Hypertext Transfer Protocol. vii, 12–14, 27, 36

HUD Heads up Display. 4

ICMP Internet Control Message Protocol. 34, 36

IoT Internet of Things. ii, 1, 2, 6, 9, 10, 12–19, 21–23, 26, 27, 29, 31–33, 36, 48–51

IP Internet Protocol. 9, 23, 28, 29

LAN Local Area Network. 9

LED Light Emitting Diode. 9, 10, 22

LiDAR Light Detection and Ranging. 1

M2M Machine to Machine. 12

MAC Medium Access Control. 27, 30

MitM Man-in-the-middle. 16, 17

MQTT Message Queuing Telemetry Transport. ii, vii, 2, 9–17, 19, 22, 24, 27, 29–34, 36–41, 43–45, 48, 50, 51

MR Mixed Reality. 5, 7

NTP Network Time Protocol. 40

OASIS Organization for the Advancement of Structured Information Standards. 13, 14

PC Personal Computer. 17, 21, 31

PICO Population, Intervention, Comparison, Outcome. 2

QoS Quality of Service. vii–ix, 16, 30, 41, 43–48

QUIC Quick UDP Internet Connections. 14

REST Representational State Transfer. vii, 10–13, 27, 29, 30

RFID Radio-frequency identification. 9, 10

RTOS Real Time Operating System. 27

RTT Round Trip Time. 34, 36

SCADA Supervisory Control and Data Acquisition. 14

SOAP Simple Object Access Protocol. 12

TCP Transmission Control Protocol. 14, 34, 36, 48

TLS Transport Layer Security. 17

UDP User Datagram Protocol. 13, 28

UID Unique Identifier. 30

URI Universal Resource Identifier. 13

VM Virtual Machine. vii, 33–35

VR Virtual Reality. vii, 1, 5, 7, 8, 17, 22

XR Extended Reality. ii, 1, 2, 4, 5, 7, 10, 11, 20, 22, 50

List of Figures

List of Tables

1 Introduction

Throughout the last decades a lot of work has been done in the emerging area of creative applications. These interesting pieces of software power art exhibits, interactive experiences in public spaces, magic mirrors in stores, Extended Reality (XR) experiences, and many other expressions of one's creativity through software. (Silva & Teixeira, 2022)

More recently with the rise of smartphones and the democratization of technology, specially mobile technologies, many new applications started implementing Augmented Reality (AR) as a part of their experience. (Alsop, 2023) With the dissemination of this early example of an immersive technology in society many developers started exploring the capabilities of the technology itself and started discovering its raw potential and possibilities in the new era of hyper connectivity.

All of the experimentation in the field of Augmented Reality culminated in 2016 with the launch of Pokemon Go, (Clement, 2024) a milestone for the dissemination of immersive technologies and the first time that the general public had a true idea of the potential and issues brought on by this new paradigm in entertainment.

If we look even closer in time, the resurgence of Virtual Reality (VR) in recent times, specially brought on by the introduction of affordable headsets, such as the Meta Quest, (Alsop, 2022) the general public has finally been introduced to fully immersive technologies. (Rubin, 2014) Even thought many leading players in this space existed before, such as the HTC VIVE (Machkovech, 2016) and the Oculus Rift (Gleasure & Feller, 2016), the drop in average price still brings with it the acceptance of many stakeholders to fund endeavors to push the boundaries of such technologies in creative ways both for private and public consumption, such examples will be discussed later in this document.

Even though many innovative creative applications have been designed for these technologies, there's still one key aspect that's still missing even in regards to fully immersive technologies: A direct connection between virtual objects in the game or experience and the real world (You et al., 2018).

1.1 Research objectives

The proposal of this thesis is the creation of a framework that can support the next generation of creative applications that are ready to use all the innovations from the last decades in the field of Extended Reality together with the possibilities brought on by integrating Internet of Things (IoT) devices in their experiences to bridge the connection between the virtual and the physical worlds.

The aim of the proposed framework is mostly focused on supporting the next generation of creative applications by enabling them to use external IoT devices for inputs, such as buttons mounted on surfaces, motion sensors, and Light Detection and Ranging (LiDAR) sensors, (Raj et al., 2020) as well as outputs, such as LED walls, stage lights, and fog machines for example.

Integrating such a wide variety of technologies together is not an easy feat, specially when there are many obstacles to overcome in terms of communication, architecture, and the overall paradigm shift needed to build applications with such systems in mind. (You et al., 2018) The goal of this research is to find solutions to these challenges and provide

empirical data to support the decisions made to build the proposed framework.

1.2 Research question

Finding a research topic for a work of this magnitude isn't an easy task, specially when the existing body of work is so vast and the amount of research being conducted in these emerging areas of technology is so big and changing in a very fast pace.

In order to narrow down the proposed research work and find a topic that wasn't well explored yet or had room for new solutions to be proposed the Population, Intervention, Comparison, Outcome (PICO) framework (Nishikawa-Pacher, 2022) was employed to create a research question:

Population Developers of creative applications and immersive experiences.

Intervention Explore the applicability of MQTT as a protocol for communication between game engines and IoT devices.

Comparison How does MQTT compare with other protocols (HTTP, CoAP, custom) used to solve the same communication issue?

Outcome How efficient and responsive is MQTT for our application and how can we make its integration into a game engine as smooth as possible.

Using the PICO framework helped better understand the research area, the proposed topic and the proposed solution. By analyzing the structure create by the PICO framework the following research question was developed: *How can a software framework be built around MQTT to be used as the backbone of creative applications and immersive experiences in order to facilitate the integration between IoT devices and game engines?*

1.3 Document structure

This document will be structured in such a way to guide readers through the research process used to develop the work proposed, a software framework for bridging the field of Internet of Things with the field of Extended Reality and creative applications.

The initial phase of the research and problem definition were detailed in this introduction, contextualizing the research topic, its field of study, and providing an overview of the objective proposed to help solve the problem exposed in the research question.

The introduction will be followed by a review of the research fields encompassed in this work, mainly the areas related to Internet of Things, IoT communication protocols, creative applications, and Extended Reality, as well as research done to find works similar to the proposal in order to contextualize and better narrow down the focus of this work.

An important section of methodologies will describe in detail the research methodologies used to validate the proposed solution to the research question, as well as explain the methodology used in the conducted tests, and the explanation behind the decisions taken along this research work.

The researched solution to the research question will be proposed in a great detail, exposing concepts and solutions that may be useful in many other applications related to the IoT field as well as the creative applications and Extended Reality industries, allowing any

researcher to replicate and test the proposed solution, as well as use it to solve problems in other areas of study.

Due to the nature of the protocol used as the basis of the proposed framework and the lack of data in this area, some performance tests were conducted in order to evaluate its applicability and viability as a solution in the proposed framework.

The research is finally concluded, the results from the proposal are analyzed, and future work in this field of research is proposed in order to advance its applicability and test its applications in order areas.

2 A review of applications and related works

In order to start designing the proposed framework it's important to understand some key concepts, research what other academics are presenting related to the areas of study as well as the innovations being proposed both by the industry and the academic world.

2.1 Extended Reality: An overview

Much of the development in recent times, when looking at creative applications, has been focused around Extended Reality, largely due to the availability and democratization of such technologies and many technological advancements brought on by the big players in the industry, enabling the creation of ever more impressive experiences. (Alsop, 2023)

The term Extended Reality is an umbrella that can be applied to any technology that aims to either extend the physical reality, such as overlaying virtual objects on the physical space, or replace it entirely, such as the usage of head mounted displays. (Rauschnabel et al., 2022)

Given its central role in recent developments, it's important to better understand these technologies and provide a background of their opportunities in this space.

2.1.1 Augmented Reality

The primordial Extended Reality technology was first developed by researcher Sutherland in 1968, in their article "A Head-Mounted Three Dimensional Display" (Sutherland, 1968), the author describes a system which was capable of overlaying images generated by a computer over a pair of clear glasses mounted on a system over the user's head. From there many other applications were found for this technology ranging from Heads up Display (HUD) with overlays for aircraft to live video production overlays and mobile video with virtual 3D objects. (Rosenberg, 2022)

These days most of the applications related to AR revolve around placing 3D virtual objects over a live image using a combination of sensors and algorithms. (Perera et al., 2014) This specific use case is mainly due to the ubiquity of smartphones and the democratization of dedicated hardware AR processing units in a wider range of mobile phones. (Alsop, 2024b)

One of the most often cited examples of Augmented Reality being used by the broad public is the introduction of the game Pokemon Go (Webster, 2017) where players are invited to go outside and play a game that's overlaid on top of physical locations around the world, but another great example of this technology being used in a much more subtle way, and sometimes unnoticed by its users, is in advertisement using magic mirrors, (Xue et al., 2022) where a screen is used with a camera to overlay objects on a reflective surface, creating the illusion that the objects actually exist in the physical environment.

Another important example of AR that often goes unnoticed is in video production, where many technologies exist that can overlay digital assets into a live video capture, similar to the way mobile phone applications work. Examples of the usage of AR technology to enhance media can be seen in many places around the video production industries, from the production and visual effects of movies to the enhancement of news studios or live sports events. (Zero Density, n.d.)

2.1.2 Virtual Reality

The promise of transporting humans to a fully immersive digital environment was always a goal of many researchers, from Cave Automatic Virtual Environment (CAVE) systems, (Cruz-Neira et al., 1992) to the modern Head Mounted Display (HMD), many systems have been tried and tested. Many iterations on the idea of full immersion have been researched, but it all culminated in the current iteration of fully immersive HMDs due to their financial viability and ease of installation and use by users. (Alsop, 2024a)

The goal of VR is to fully immerse the user in an environment, transporting their senses to the digital realm and taking most of the control possible to create the illusion of being in a completely different space. This approach is the polar opposite of AR technology, trying as much as possible to distance the user from the surrounding environment. (Burdea & Coiffet, 2003)

This approach has the advantage, from the developer's perspective, of taking full control of the experience, allowing for much simpler development of the experience due to not relying on any external inputs other than the motion sensors in the headset and some times trackers for controllers and other objects. (Rauschnabel et al., 2022)

2.1.3 Mixed Reality

Due to the introduction of the next generation of XR headsets, from companies such as Apple with its Vision Pro, (Apple, 2023) and Meta with the Quest 3, (Pierce, 2023), which focused heavily on mixing VR and AR content, the creation of a new category named Mixed Reality was required. (Aiersilan, 2023)

This new category of immersive devices operate at the intersection between AR content, with overlays of applications and 3D objects or scenery over a live high speed video feed, as well as fully immersive environments and experiences where the user is completely immersed in the content. Some devices, such as the Vision Pro, can seamlessly transition from VR to AR and vice-versa on the same application with a linear immersion control. (Apple, 2024)

One important breakthrough of this technology is that the applications currently being developed are aware of the physical space through the use of a range of depth sensors, allowing applications to blend into the existing environment dynamically and with accuracy. (Fischer et al., 2023)

2.2 Digital twins

The concept of a digital twin may become ever more present due to the future need of replicating physical environments in fully immersive experiences, or allowing parts of a virtual experience to be shared with other users that are not fully immersed.

A digital twin is a virtual representation of a physical object. The idea behind this concept is to characterize systems where environmental elements must be replicated in a virtual realm, in order to improve a simulation, or to better immerse a user into an experience, thus providing a bridge between the real and the virtual. (VanDerHorn & Mahadevan, 2021)

There are many applications of such technology, ranging from industrial, where digital

twins of physical systems can be created to better visualize and monitor a system (Moi et al., 2020) or to fully simulate its characteristics in a controlled and safe environment, (Haag & Anderl, 2018) but these twins can also be used for entertainment, where for example physical objects can be replicated to allow a player to play a physical board game in-person with an opponent playing a digital version of the same game in completely different location. (Han et al., 2023)

Because of the generalized definition of a digital twin, many examples can fit into its umbrella, although the term is usually applied to situations where environmental objects such as trees, cars, tables or chairs, and their positions are replicated in a virtual world, many times a digital twin is a lot more complex, such as a virtual representation of a physical sensor or device, where the entire behavior of the object is simulated and replicated in the virtual environment. (Liu et al., 2021)

2.3 Creative applications and extended reality in production

Creative applications, as related to the field of study and purpose, are a category of interactive media that with strong focus on immersion, entertainment, and or innovation. (Sourin, 2017) Examples of such applications can include games, art installations (interactive and non-interactive), and interactive experiences or immersive experiences both in public or private venues. Everywhere there's someone's creativity being expressed through software can be thought of as a creative application. (Connor, 2020)

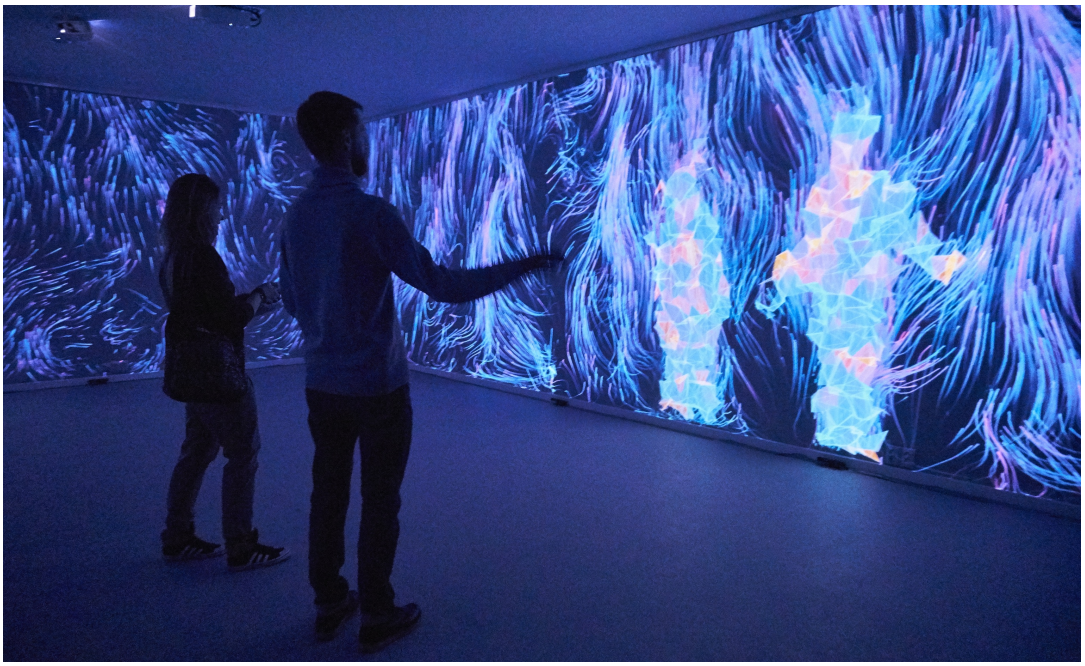

Figure 1: Example of an interactive art exhibit. (Sodazot, 2015)

Art installations when software is incorporated for interaction purposes or not, are a great example of the target use case. Artists has redefined themselves in recent times by adopting new technologies to express their work, largely due to the availability and affordability of many solutions, creating expositions that were unconceivable in the past. IoT sensors, multiple screens, video mapping, and interactive media are commonplace in most exhibitions around the world. (Li & Huang, 2023)

Another great example of creative applications are immersive and interactive experiences. Many museums, public venues, and events are now offering exclusive experiences to their customers in order to broaden their audience or to create a more engaging and compelling experience to visitors, opening up a substantial market with many opportunities to explore, given their acceptance to innovation and technology. (Baradaran Rahimi et al., 2022)

Figure 2: Example of a creative application from the Smithsonian's National Museum of Natural History Bone Hall. (Smithsonian, n.d.)

It's important to note that the field of Extended Reality, which encompasses Augmented Reality, Virtual Reality, and Mixed Reality, due to their innovative and immersive experiences provided are considered an integral part of the creative application's field and currently represent most of the commercial interest in this area. Due to the recent availability of XR devices such as Microsoft's HoloLens and Tilt Five AR glasses, HTC VIVE and Meta Quest 2 VR headsets, and more recently the Meta Quest 3 and Apple Vision Pro Mixed Reality (MR) headsets, many creators are targeting these systems for their creative applications, which is to be expected given the large unexplored potential of new technologies. (Wedel et al., 2020)

Some noteworthy examples from the XR space include educational applications aimed at providing a much more engaging experience to topics which were generally explained in a much more traditional way, capturing the attention of students used to the general availability of interactive media brought by smartphones and tables, and enriching their learning experience. (Kavanagh et al., 2017)

Figure 3: Example of Augmented Reality being used in the classroom. (ClassVR, 2022)

More recently it's noticeable the emergence of interactive experiences for consumption at home, where users are transported to a fully immersive environment crafted to enhance an otherwise traditional experience, such as Radiohead's Kid A Mnesia Exhibition, (Radiohead, 2021) which elevates the hearing experience of their music album Kid A Mnesia to a whole new level using Virtual Reality technology.

Figure 4: Radiohead's Kid A Mnesia Exhibit Virtual Reality experience. (Peters, 2021)

Another type of interactive experience that has been growing in recent times are Augmented Reality games such as Pokemon Go, but also interactive displays in public spaces such as magic mirrors that are gaining traction in the retail space, allowing shoppers

to have an experience akin to online shopping, with a vast selection of browsable items, which can be overlaid on their body for an enhanced try-on experience. (Memomi, n.d.)

2.4 Related works

Although the proposed framework for solving the integration of IoT devices and a game engine may be novel, it's still building on top of the work and research of others that have contributed to the academic corpus. Their researched topics helped us narrow down the research focus and sparked the idea for the proposed solution.

2.4.1 UNITY-Things

In their article "UNITY-Things", Svanæs et al. (Svanæs et al., 2021) propose a framework where Internet of Things devices, more specifically ESP8266 development boards, were connected to the Unity game engine using the MQTT protocol in order to extend the input and output capabilities of a game.

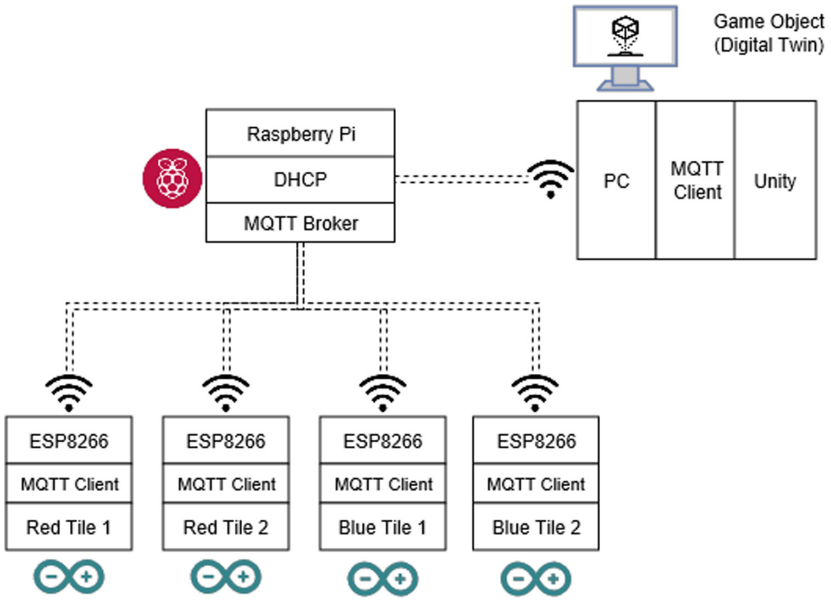

Figure 5: UNITY-Things system architecture. (Svanæs et al., 2021)

Their proposal had the IoT devices talking to the Unity game via a Raspberry Pi 3 computer that was providing a wireless Local Area Network (LAN) access point for all involved devices, as well as a Dynamic Host Configuration Protocol (DHCP) server, in order to distribute Internet Protocol (IP) addresses automatically between clients, and an MQTT broker.

A Unity library was developed to allow game developers to more easily associate virtual game objects to the actual physical devices, thus creating a digital twin of the physical object.

A proof of concept empirical test of the system was created where ESP8266 based modules with Light Emitting Diodes (LEDs), used as outputs for the game, and Radio-frequency identification (RFID) readers, used as input for the game. These modules were programmed with the Arduino based part of their proposed framework.

Figure 6: UNITY-Things example implementation of a game. (Svanæs et al., 2021)

The IoT modules were programmed to subscribe to topics where Unity would be capable of controlling the color of the LEDs on the module, from the Unity side of the digital twin they had to configure the digital game object to subscribe to the topic where the IoT module would publish the information gathered from the RFID reader and act accordingly in the virtual realm.

This article most importantly showcases the possibilities created by integrating IoT modules with games and interactive experiences built with the Unity game engine while using the MQTT protocol to build the bridge between the two technologies.

2.4.2 IoT-ready XR-WebApps

Fleck et al. in their article titled "Creating IoT-ready XR-WebApps with Unity3D" (Fleck et al., 2020) explore the possibilities brought on by integrating MQTT and web technologies in order to create next generation Extended Reality web applications that can react in real-time to changes brought by the application's server or IoT sensors.

By using web technologies to develop web applications with XR devices in mind the researchers were able to showcase the creation of complex user interfaces with HTML, CSS, and Javascript, just like any regular web application, that can easily be rendered in XR environments using the Unity game engine with the help of third-party libraries.

Their web application leveraged a complex server-side system to showcase the use of MQTT for data collection from sensors that were stored by the server acting as a subscriber, but also streamed in real-time to the XR client also acting as a subscriber to the broker for the same MQTT topic.

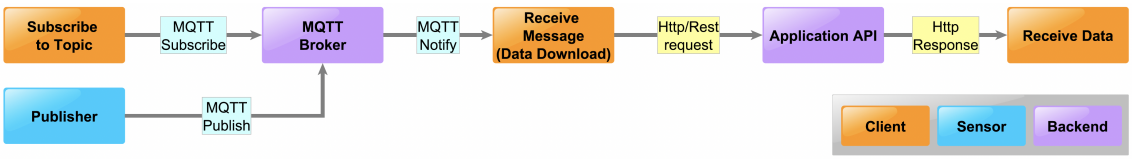

Figure 7: Diagram showing how their proposal can replace REST API pooling. (Fleck et al., 2020)

It's important to note that their system wasn't entirely based on MQTT as most of the web application used the common REST API architecture in order to streamline the application development. MQTT was only used for real-time information and to provide instant notifications to the application for available updates.

Although not using MQTT directly to exchange information between the server and the XR application the authors showcase how the protocol can be used to replace REST API pooling with an event-driven system that notifies clients when to fetch updated data from the server, saving on resources that are at a premium on many of these XR devices.

2.4.3 Vehicle Digital Twin

With the recent developments in vehicle automation it becomes crucial to be able to monitor and simulate these highly-connected vehicles. The creation of a digital twin which researchers can use monitor and replay scenarios becomes an appealing proposition. (Wang et al., 2021)

In "Digital Twin Simulation of Connected and Automated Vehicles with the Unity Game Engine" Wang et al. propose their system to control and monitor such vehicles using the Unity game engine, MQTT and many other cloud technologies.

The proposed system uses the MQTT protocol to relay information gathered by onboard car sensors as well as external sensors placed in the car in order to help recreating the physical world in the virtual simulation. The protocol allows the car to send its data as well as be controlled by the digital twin in the virtual simulation.

Figure 8: Example of the interaction with the system controlling a digital twin. (Wang et al., 2021)

This research not only demonstrates the efficiency of the MQTT protocol and how it can be used in high-speed applications, since latency in the communication can negatively impact the simulations that are being produced by external systems, but also demonstrates the flexibility of the protocol in terms of the data that can be exchanged as well as its interoperability with other software systems.

2.5 Communication protocols

Being largely tied to the engineering space, there are many communication protocols available for embedded devices. Some are intended to be used internally by the devices in

order to communicate with their onboard peripherals, such as I^2C, SPI, and UART, while others are intended to provide communication with other devices such as USB, DMX-512, CAN, 802.11, and LoRa. For the purposes of the proposed work, there won't be a focus on these low-level, physical layer, protocols with very limited abstraction, instead the focus will be on protocols used by IoT devices that are already part of a network capable of communicating over a network.

In this segment some protocols clearly stand out due to their ubiquity in the industry and academic literature: HTTP as the *lingua franca* of the internet, MQTT as a lightweight protocol designed specifically for IoT applications, and CoAP as a modern lightweight protocol also specifically built for IoT.

2.5.1 Hypertext Transfer Protocol (HTTP)

The venerable HTTP protocol has been a staple of the internet ever since its introduction in 1991. It provides a simple, text-based, request-response, application interface to request and update hypertext documents, using a combination of *verbs* and *headers* to control the information exchange.

Although HTTP is still widely used to provide hypertext documents in the form of web-pages, it was the introduction of techniques such as REST (Fielding, 2000) and Simple Object Access Protocol (SOAP) (W3C, 2000) that turned it into an entire API for web services, allowing clients to exchange information with web servers using a Machine to Machine (M2M) language that was widely accepted by web technologies.

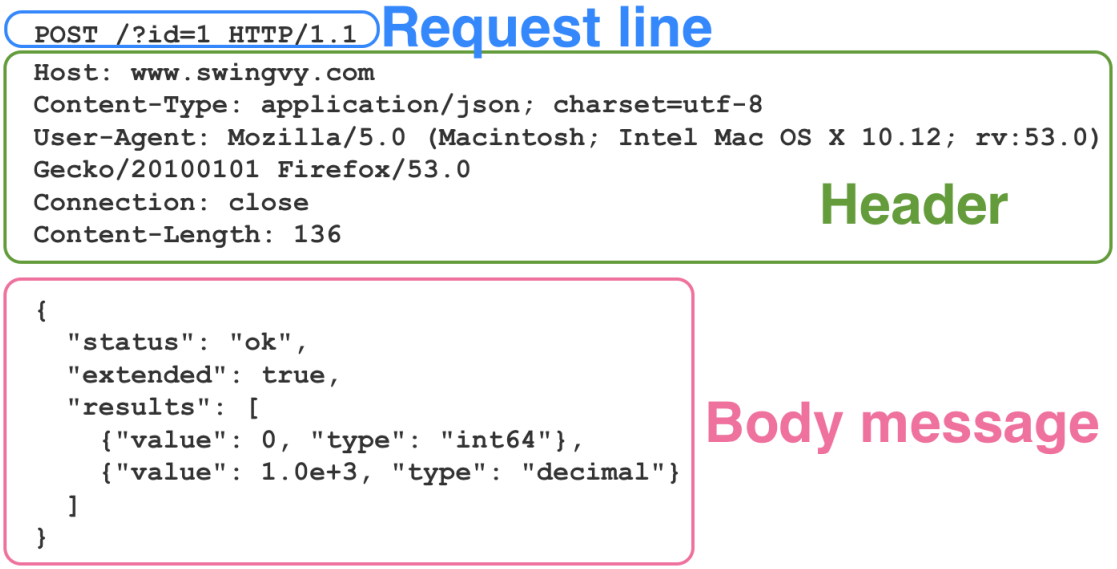

Figure 9: Example of an HTTP request to a REST API. (Notes, 2021)

By implementing a RESTful application a client can directly interact with the underlying service and allows for simple and effective solution that's very simple to develop. Since the interface is standard web technologies a plethora of clients can use the same underlying service, such as websites, mobile applications, and IoT devices.

Although the protocol itself is designed to be extremely simple and easily extensible, due to its text-based nature and unpredictability, it can be a difficult protocol to parse or

construct messages for, specially on highly constrained devices such as some low-power IoT compute modules. Even though many IoT solutions still use it for communication due to its ubiquitous nature and familiarity. (Naik, 2017)

2.5.2 Constrained Application Protocol (CoAP)

The CoAP protocol was developed by the IETF Constrained RESTful Environments Working Group in 2011 with the goal of providing a modern alternative to MQTT and other protocols specific for resource constrained applications that was easily translatable to and from HTTP. The protocol is binary-based and supports both request/response and resource/observe, its own variant of publish/subscribe, architectures. (Shelby et al., 2014)

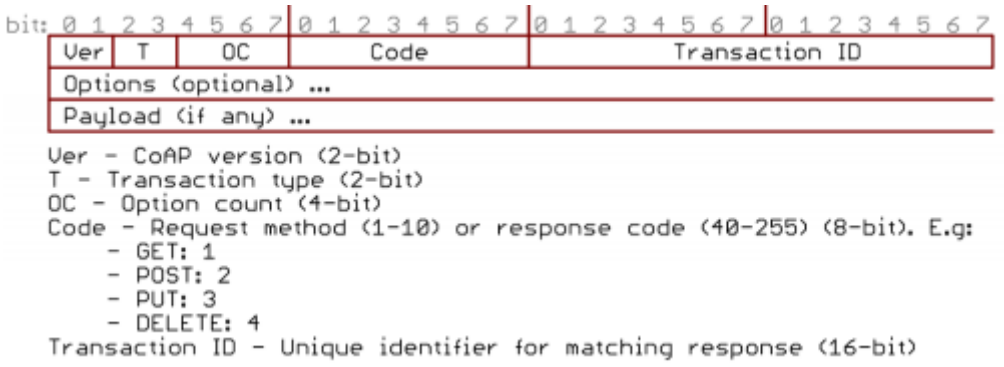

Figure 10: CoAP packet structure. (Dallinger, 2023)

The protocol was developed with the intention of interoperability with HTTP and for this reason contains traces of it and MQTT combined in order to create its solution, because of this, the topic structure was substituted for an Universal Resource Identifier (URI) based addressing approach. Although, different from other approaches the authors of the specification decided to use User Datagram Protocol (UDP) for as the underlying communication mechanism. (Bandyopadhyay & Bhattacharyya, 2013)

Even with its modern feature set and interoperability with HTTP, largely due to familiarity and a proven track record, this protocol wasn't able to take much of MQTT's market share, specially when version 5 of the MQTT protocol addresses many of its shortcomings that were hailed as benefits of CoAP. (Blom, 2015)

2.6 MQTT: An overview

Given the large body of work in this field revolves around the MQTT protocol, with numerous examples of its usage and benefits in this field of research, many of which relate closely to the problem aimed at solving, it was imperative that more research and contextualization be done around this protocol, its applications, highlight where it's used and why, its limitations, and any potential challenges.

2.6.1 What is MQTT?

According to MQTT's official website: (MQTT.org, 2022a)

> MQTT is an Organization for the Advancement of Structured Information

> Standards (OASIS) standard messaging protocol for the Internet of Things (IoT). It's designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

Just from this almost mission statement it's clear that this protocol was developed and optimized for IoT applications, not general purpose data transferring (ala HTTP), meaning it will be limited in its scope and usage. This can be viewed as a disadvantage to some, but in this case it's important that the protocol can be easily implemented in resource-constrained devices and also have a very small footprint in order to reduce any unnecessary latency.

2.6.2 History

As with almost every well known protocol, it was developed by one of the big technology companies, in this case IBM, in 1999 by Arlen Nipper and Andy Stanford-Clark (MQTT.org, 2015) to control the newly installed Supervisory Control and Data Acquisition (SCADA) control systems used in the Phillips Conoco oil pipeline over satellite. ("IBM Podcast Transcript", 2011)

Initially this protocol was proprietary and marketed by IBM as MQSeries as a message queuing (hence the name) system. In 2010 IBM released the version 3.1 of the specification royalty free, (IBM & Eurotech, 2010) already bearing the name MQ Telemetry Transport, letting the general public finally take a peek at this revolutionary protocol. In 2013 IBM submitted the 3.1 revision of the specification to OASIS which published it as an open protocol in 2014 with the name being officially changed to MQTT and the version to 3.1.1. (Gupta, 2015)

In 2019 OASIS released the 5.0 specification of the protocol, (OASIS, 2019) skipping version 4, with many new features. (MQTT.org, 2018)

2.6.3 An overview of the protocol

The MQTT protocol typically relies on Transmission Control Protocol (TCP) for its layer 3 given that it requires a transport protocol that provides ordered, lossless, and bi-directional connections, but the much newer Quick UDP Internet Connections (QUIC) can also be used if all of the entities involved support the 5.0 version of the specification, which added such possibility. (EMQX, 2022) By default uses port 1883 for unencrypted traffic and 8883 for encrypted.

In the specification only 2 types of network entities are defined:

Broker Usually a piece of server software that receives the messages published by the clients and stores and routes their messages to other clients that subscribe to the same topic.

Clients IoT devices or any sort of internet connected device that may receive or send messages to the same broker.

As the workflow diagram demonstrates, messages are sent to the `broker` with a specific

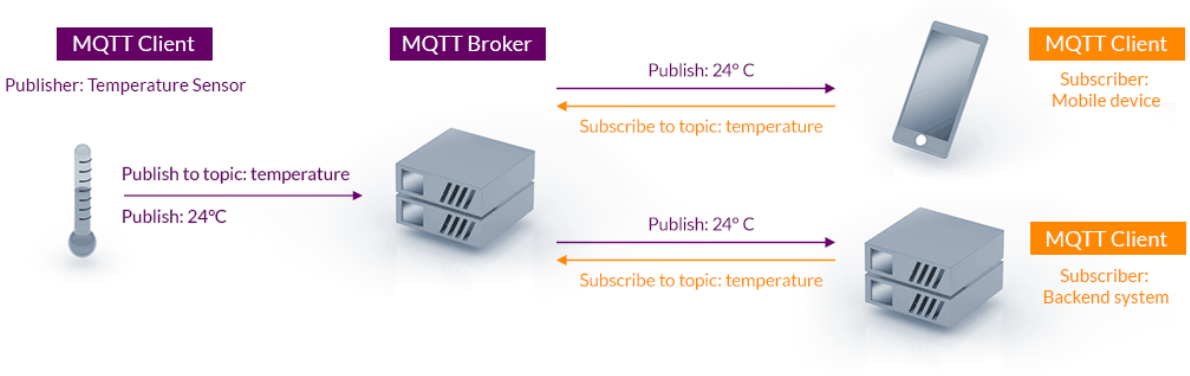

Figure 11: Example of the MQTT workflow.

`topic` by a `client`, usually an IoT device like a temperature sensor for example, this is called a `publish`. Other clients such as a smartphone application, another server, or a control system can `subscribe` to `topics` and receive updates as these messages arrive to the `broker`. The system is meant to be extremely simple and require minimal amount of overhead from all parties making it very easy for applications to communicate and interact with sensors and other remote IoT devices.

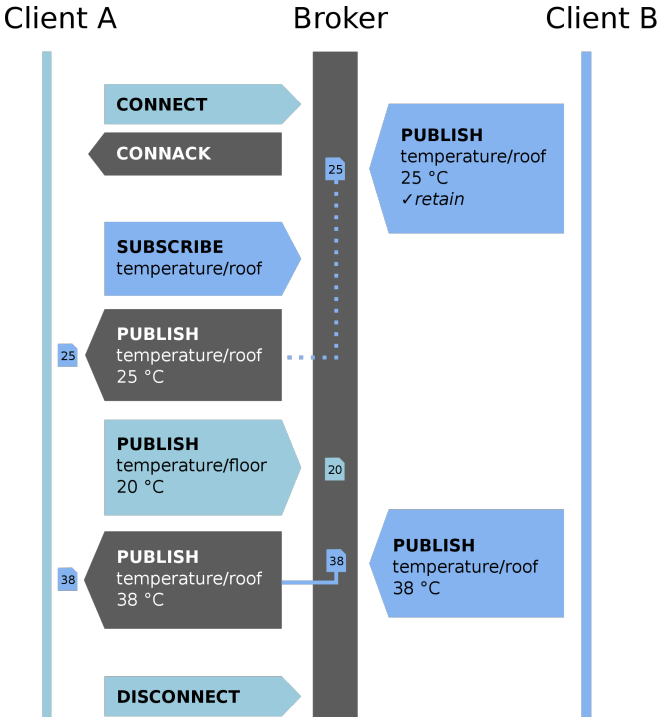

Figure 12: Example of some simple transactions using the MQTT protocol. (Eugster, 2018)

Since MQTT is meant to require very little bandwidth and processing power from its clients a message can be as little as 2 bytes of data and scale all the way up to 256 megabytes if required. A collection of 14 message types can be used to perform all of the operations defined in the specification.

Due to its roots in bandwidth efficiency the connection credentials are sent in plain text and don't include any provisions for security or an authentication mechanism. The only

"protection" offered by the specification is to connect to the `broker` using TLS to encrypt the connection and protect the information being transferred from Man-in-the-middle attacks.

2.6.4 Why is MQTT used?

MQTT has some clear advantages in the system architecture, being extremely simple to implement even on extremely constrained hardware, which is usually the case for IoT devices, which might be running on batteries and have extremely limited microcontrollers as their main processing units.

Apart from the clear lightweightness of the protocol, its hierarchical topics model integrates seamlessly and perfectly models the organization of the real world in which IoT sensors and devices might be encountered, allowing for an almost one-to-one relationship between a topic and a physical property, which is far better than other kinds of tagging or addressing systems based on IDs.

Due to the centralized nature of the protocol by the means of a broker it makes the retrieval and delivery of a single message to many clients, which is usually the case when IoT is considered, very simple and efficient.

Another important point of note is the fact that the protocol allows for three levels of QoS, ensuring the reliability of message deliveries. This coupled with the persistent session nature of the protocol allows for a reliable transfer of data even when the medium isn't optimal, such as radio, satellite or cellular networks.

2.6.5 Where is MQTT used?

By design, ever since its inception, MQTT was a protocol destined to be used as a kind of glue logic to piece together IoT sensors and devices and larger computational systems, because of this nature it'll always be seen in places where distributed aggregation of physical data is required, which typically means it'll be heavily used in a residential setting for automation or in an industrial setting for data gathering or automation.

An amazing list of use cases and areas can be consulted on MQTT's official website (MQTT.org, 2022b), providing a glimpse of its wide range of uses throughout all of the major industry areas.

2.6.6 Message Queuing: The limitation

Even though the name of the protocol states that it's a message queuing system, it's not truly a message queue, given the fact that, unless a message sets the QoS level to a value greater than 0, no messages are going to be queued by the broker. As messages are published they'll be sent to subscribed clients in a First in first out (FIFO) fashion.

Due to this limitation and the fact that usually there is an interest in storing the typical time-series data that's collected by these IoT platforms, data storage systems are usually directly integrated by the broker software or indirectly integrated by means of a client software.

An important aspect of this no-queuing system is that it fits quite well into this proposal where near-real-time data can be useful as an input method for an interactive experience.

2.6.7 Security: MQTT's biggest challenge

Because of its history and focus on being bandwidth efficient the protocol never had a focus on security, which wasn't a big deal back in 1999 when it was first developed, but leaving such implementation details out, specially in the modern era where high computing power and purpose-built malware is a reality is a major oversight.

The only security feature that was baked into the specification was the ability to encrypt an MQTT session over Transport Layer Security (TLS), ensuring only the authenticity of the clients and protecting from Man-in-the-middle attacks and message snooping, but a compromised client can still send malicious messages back to a broker with no way, from the broker's perspective to know if the client has or hasn't been compromised.

One example of a recent attack on the protocol was a vulnerability in the version 3.1.1 of the protocol, which is still by far the most widely used version of the specification, where brokers can suffer slow Denial of Service (DoS) attacks by abusing the server's timeout/Keep-Alive value and exhausting its available connections. (Vaccari et al., 2020)

2.7 HTC VIVE Ultimate Tracker: A goal

It's no secret in the immersive experiences industry that HTC with its VIVE line of Virtual Reality headsets is a market leader in public space VR applications, largely due to its initial headset release, technical support, and devices and peripherals largely targeted at this specific industry. (Heater, 2024)

Due to its prevalence in the creative industries, going as far as sponsoring the creation of some of them themselves, (Gepp, 2017) the company is constantly looking for ways to differentiate themselves from its competitors, which usually target the consumer market with more affordable solutions, and bid on providing more utility to their commercial partners, in 2023, they unveiled to the industry the VIVE Ultimate Tracker, (Porter, 2023) a set of small tracking devices that can be used for full body tracking in VR, enabling even more immersive experiences, but also general object tracking.

The introduction of this product is a direct response to their market searching for novel ways to create even more immersive experiences and blend the physical and virtual worlds even further, something that's intimately aligned with the vision for the proposed framework.

Although their device has a lot of potential mostly because its supported by their own VR headsets natively and third-party ones when connected to a Personal Computer (PC) using a dongle, and is considered a breakthrough in VR-assisting technology (Porter, 2023), it's usage is still limited due to being a closed, proprietary, platform from its hardware, to the protocols and the software required for it to function, it cannot be extended or its tools be used by other systems that aim to expand its capabilities.

If a developer were to integrate VIVE's system into their application but require another IoT solution to provide an intended capability to their experience, they would have to integrate yet another system into their project, adding complexity, maintenance cost, and possibly incurring a performance penalty, when an extensible system could've easily solved their problem.

Figure 13: HTC VIVE Ultimate Tracker 3.0 with body straps. (Porter, 2023)

The *Ultimate Tracker* can be seen as a goal for the proposed framework to solve for. It should be aimed at bringing together the technologies needed for devices such as these to be easily integrated into immersive applications and able to be extended to accommodate a large variety of current and future use cases in bridging the gap between the physical and the virtual worlds with IoT technologies.

3 Methodology

In order make this a viable proposal and create a framework that can continue to be used in the future in a sustainable manner, that allows for future applications to exist, and take advantage of advances in technologies in the areas that we'll be touching, the system will be comprised of a series of industry-standard tools, including protocols, development environments, and hardware as they are more likely to still be in active use by professionals in the areas of interest related to this research for at least the foreseeable future.

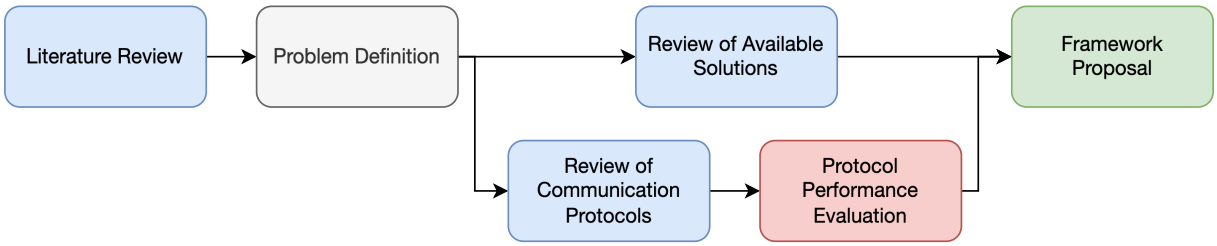

Figure 14: Diagram of the methodology applied in this proposal.

In order for the IoT modules and the client applications to communicate it's of utmost importance that a standardized protocol is used to ensure that the proposal has a sustainable development environment for future generations of the system. It's also important to use a protocol that has the least amount of overhead, so that it can be easily implemented in the constrained environment typically encountered in embedded devices, such as the ones we'll be using in the modules, without compromising the flexibility in terms of the information that can be transmitted. The industry-standard protocols for IoT device communications are plenty (Naik, 2017) but the most widely used and efficient one is MQTT (Mishra & Kertesz, 2020).

Given the large body of work related to the usage of IoT devices in applications largely being focused on MQTT as the underlying communication protocol for such tasks, the proposal will also focus on this protocol for the direct communication with such devices, given the proven track record in this field and low overhead incurred by the protocol itself. (Mishra & Kertesz, 2020)

Although works on this area are heavily focused on MQTT being a lightweight and widely used protocol, as can be seen in the related works review, they are usually also focused on integrating sensors and other devices that do not require fast refresh rates in order to function or integrate well into their proposed systems. The specific application for these devices will of course also involve these traditional IoT devices as general input and output, although the proposal will mainly revolve around the less explored area of using them as input devices or game controllers, thus requiring extremely low latency figures, usually below 50 ms in order to provide an acceptable experience for users. (Long & Gutwin, 2019)

In order to confirm the hypothesis that MQTT can in fact be a viable communication protocol for such an use case a series of benchmarking tests should be conducted in order to infer the latency introduced by the protocol itself, its related pieces such as the broker software, and the client-side libraries needed in order to implement it. If the viability of the protocol is confirmed at this stage, future works can be devised to ascertain its practical capabilities outside the limited scope of this proposal and other use cases in creative applications.

Since the proposed framework is heavily leaned on the development of immersive experiences, games, and related creative applications, these are usually developed using specialized tools that differ from traditional application development called game engines. (Holtmann & Wernike, 2023) Although their name associates them directly with games, which are still their primary focus, these days they are commonly used in video production and related industries due to their exceptional high fidelity 3D capabilities. (Cremona & Kavakli, 2023) Their name is mostly a historical artifact of to their origins as tools for writing games with. (Thomsen, 2010)

Under the hood these engines provide an abstraction layer over the operating system's primitive 3D graphics routines, bundle commonly used physics, Artificial Intelligence (AI) and computer graphics related routines, and provide an intuitive and easy to work with environment for creating 3D environments and programming them. (Andrade, 2015)

Of the most popular games engines in the market (Doucet & Pecorella, 2021) two stand out: Unreal Engine (Epic Games, n.d.) and Unity (Unity Technologies, n.d.). Unreal Engine is mostly used by AAA games (Mathews & Wearn, 2016) and the multimedia industry (Cremona & Kavakli, 2023), with a very steep learning curve and not a lot of developers working on it outside of these industries, although it has been gaining some traction in the XR market. Unity is mostly used by indie games (Mathews & Wearn, 2016), is very easy for beginners to learn and has a major community of hobbyists and industry developers in many diverse areas. Unity also has a major foothold in the XR market, including many immersive experiences found in public spaces around the world (Komianos, 2022) and a vast collection of successful commercial games.

In the interest of limiting the scope of this work to the actual development of the framework and testing its viability, a single game engine had to be chosen to represent their role in the proposed framework and to underpin the tests. The Unity game engine was chosen given its market share in the gaming and creative application space as well as its appearance in many of the researched documents. (Hussain et al., 2020)

4 The proposed framework

There are many great examples of creative applications using the power of the integration between game frameworks and Internet of Things devices being used in public venues around the world on a variety of subjects and artistic endeavors, but most of them are either, using off-the-shelf peripherals that already integrate with PCs or mobile devices, and that's the reason why they were chosen, or have had to develop fully customized and bespoke solutions in order to integrate such devices into their projects. (Salim & Haque, 2015)

The aim of this research work is to provide a standardized framework that can easily be implemented both on the application side and on the embedded side, with libraries and specifications describing the workflow of the integration, allowing any programmable and low cost IoT development board, such as wireless-enabled Arduinos, Espressif modules, and others, to be programmed to directly integrate into an application built using the same framework in a seamless fashion.

It's important that, from the application's point-of-view, these external devices are integrated in a way that's not intrusive to the application's code, allowing a developer to bypass their usage if needed or in case they aren't available, ensuring they are not a requirement to the execution of the application itself, unless the developer explicitly wants so. This requirement means that the developed library must not create any mandatory event loops or force developers to implement specialized event handling methods, and must provide access to events using Unity's standard `UnityEvent` interface, (Unity Technologies, 2024) making inputs from IoT devices simply another input to an event in the application.

Another important point of the proposed framework is that an application developer with little knowledge of Internet of Things programming and circuit building can take advantage of easy-to-use hardware building block solutions such as the M5Stack (M5Stack, n.d.) or the BBC micro:bit (BBC, n.d.) to create custom solutions for their needs and be comfortable programming the firmware of such devices by using an abstraction layer where all they have to do is setup the sensor or output device they want and customize the event loop in order to send the required data to their application.

4.1 Entities

Because of the number of parts involved in the framework and the relative complexity of some possible scenarios of usage and device combinations, it's important to clear some terminology that will be used for the rest of the document to reference some integral parts of the framework.

It's understandable that some of the proposed names for entities in the proposed system may cause some confusion, largely due to the fact that these terms are already used in computing circles for similar, or larger encompassing, entities. Although the only way to properly address each of the entities proposed was to assign meaningful and bespoke names to each entity in order to not cause confusion between the parts of the framework itself and the context in which the referred terms are used must be retained.

4.1.1 Application

What is hereby referred to as the `application` is the actual software developed for the creative application, such as an interactive experience, game, or interactive multimedia production.

At this stage the `application` must be developed using the proposed framework's game engine, Unity, but given the public specification, any software that integrates into the framework with the same functionality will also be considered an `application`.

4.1.2 Clients

`Clients` in the proposed framework are simply the devices that are actually running the creative application or game. These can range from computers connected to other infrastructure, such as LED walls, projectors, or Virtual Reality headsets, to XR devices themselves or regular mobile devices.

These can be viewed as all devices within a system that implements the framework that are actively running the `application`, of which there can be multiple, as exemplified in Figure 15, if it's the intention of the developer, and may operate independently from each other and receive inputs from different IoT devices (`nodes`) throughout the system.

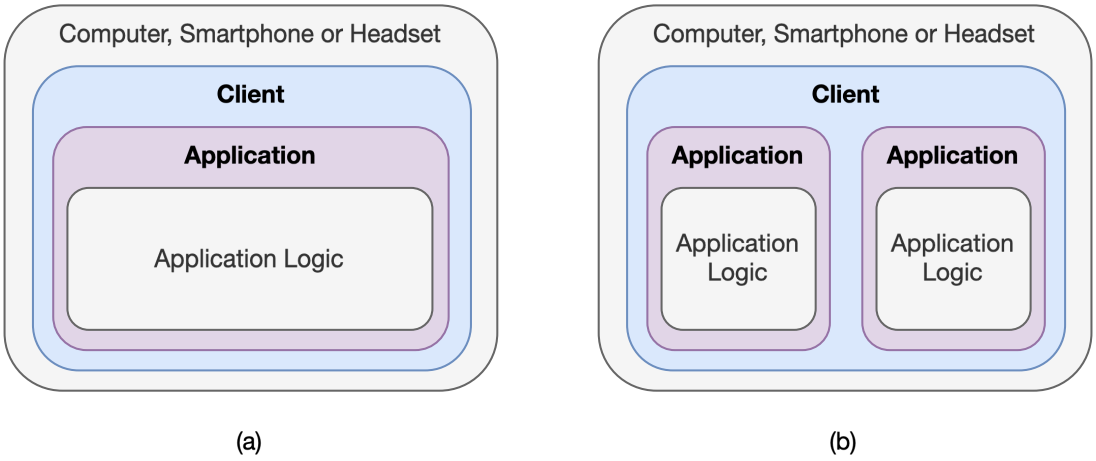

Figure 15: An example of how `clients` relate to `applications`. (a) being a `client` that runs a single `application`, and (b) being a `client` that runs multiple independent `applications`.

4.1.3 Personalities

In order to provide a bespoke name to the attributes related to the IoT devices (`nodes`) in the framework, such as temperature sensors, RGB LEDs, buttons, and motor rotations, a naming convention from the Digital Multiplex Protocol (DMX512) world was borrowed. (Wan, 2022)

`Personalities` are all the individual bits of information that can either be read or written to in a `node` by the controlling `application`. These can represent inputs or outputs on a device.

It's important to note that a `personality` refers to every single MQTT topic a `node` publishes, in case of inputs, or subscribes to, in case of outputs. For example, as can

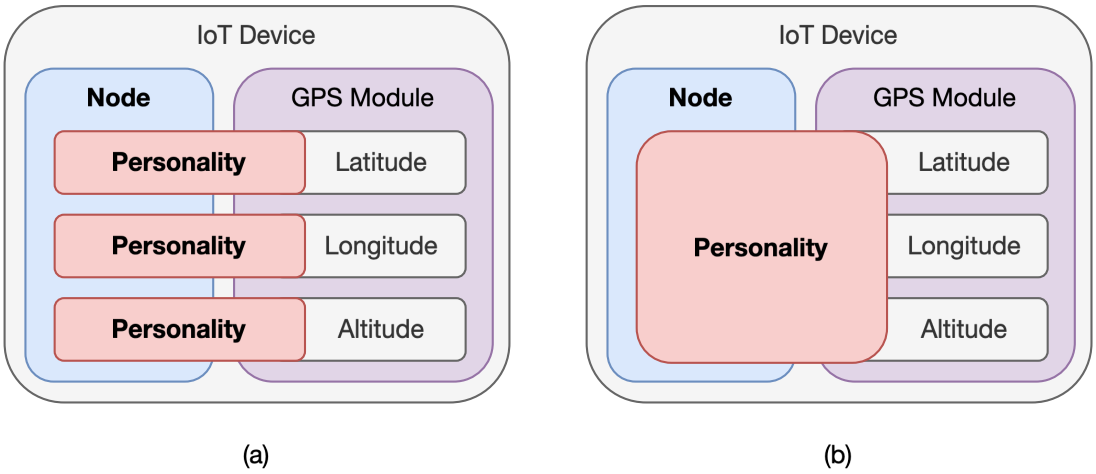

Figure 16: An example of how each `personality` can relate to (a) a single value from a module or sensor or (b) multiple values serialized into a single `personality`.

be seen in Figure 16, if a `node` in a system contains a Global Positioning System (GPS) module capable of providing the latitude, longitude, and altitude of the device, each one of those attributes can considered a single `personality` of the `node`, if the developer so desires, or a single `personality` that publishes the device's position, with all those parameters serialized, to a single *topic*, it's up to the developer of the `application` to decide.

This distinction between the `node`'s peripherals and the peripheral's properties or capabilities is intentional in order to not tie the application to a specific peripheral type that may change in the future, but instead tie it to the general capabilities of the entire `node` itself, not having to know the specificities of how that module was built at a hardware level.

4.1.4 Nodes

Every IoT module that's acting either as an input or output, or both, to the `application` running in the `client` must be classified as a `node` on the framework. The name was chosen to imply their small size and the possibility of having a large number of them deployed.

These `nodes` may contain a single `personality`, such as a light acting as a single spotlight to emphasize an area of an exhibition, or a button to activate another part of the experience, but they can also combine several `personalities`, such as an illuminated button that when pressed changes color as instructed by the `application`.

No matter the functionality of a `node` or the number of `personalities` it may contain, it'll always be categorized as a single `node` on the framework and in the `application`. The only exception to this would be devices that contain multiple radios onboard and connect to the physical network as multiple devices with different IP addresses. In such a case, each device with an associated IP address and its associated `personalities` will be treated as a single `node`, independently of its physical placement.

4.1.5 Mediator

Due to the unique characteristics of the MQTT protocol that assume the *topics* to be used for communication between devices have been predefined, in a dynamic system such as the one proposed this is not possible. A way to dynamically configure the MQTT *topics* assigned to each `node` is required, and thus the role of the `mediator` was born.

In the proposed framework only a single `mediator` must exist and be used for setting up communications between `clients` and `nodes`. This piece of software may or may not be physically installed in the same hardware as the MQTT *broker* or be running alongside the `application`.

4.1.6 Broker

The `broker`, as the name implies, refers to the MQTT *broker* that's used to relay information between `clients` and `nodes`. Due to the presence of the `mediator` it's possible for multiple `brokers` to coexist in a single network, as long as the application is programmed to take advantage of this feature.

The proposal makes the concession of having the possibility of multiple `brokers` for situations such as `client` running its own `broker` for possible latency reasons, or multiple `brokers` being used to balance the load in situations where many `nodes` may be overloading a single `broker`. Although it's believed that these configurations are not necessary and won't be prioritized in the implementation.

Although any compliant MQTT *broker* software can be used here, it's advisable to test implementations that introduce the least amount of latency possible to provide an acceptable level of performance to users.

4.2 Architecture

The framework has been architected in such a way to maximize the possibility of future expansion and provide flexibility so that applications are not limited to a small subset of possible devices to use.

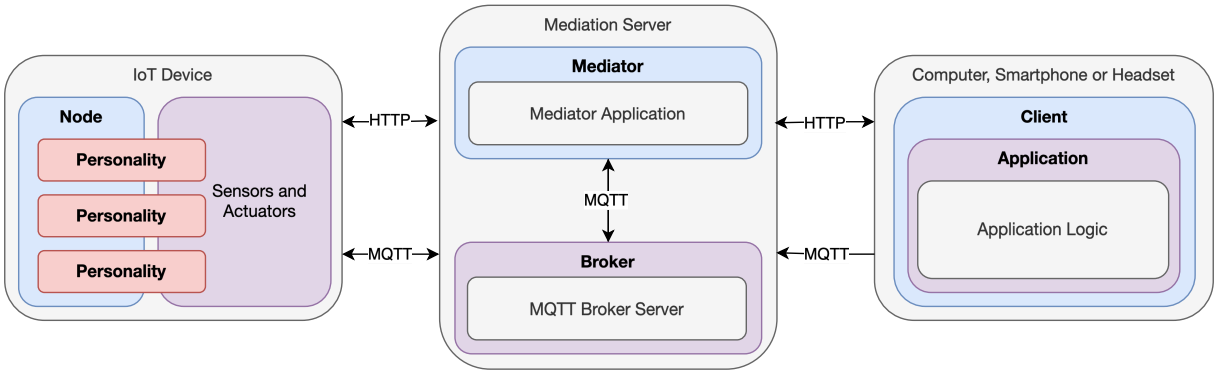

Figure 17: Architecture diagram of a system with a single `node` and a single `client` within the framework.

In order to provide a glimpse into how all these moving parts may interact within the framework, Figure 17 shows a diagram of the architecture of a system with a single `node`

and a single `client` helps to illustrate the interactions between the different parts of the framework.

This is a simple diagram with a very limited set of functionality, but it showcases an overview of every single entity in the system and how they are arranged and interconnected in a more intuitive way than just text explanations.

4.3 Unity package

In order to achieve the proposed goal of a framework for integrating non-standard peripherals as input and/or output devices for creative applications, some limitations or quirks need to be addressed in order to ensure its viability for developers.

Leveraging the opportunities and possibilities brought on by the integration of such a broad range of possible device types, the abstraction brought on by this framework should be designed to provide a painless way for developers to easily integrate them into their applications without requiring a major refactor of the code base, or depending too much on the existence of the devices for testing.

As an easier way of integrating the proposed framework into new, and existing, Unity projects, an *Asset Package* will be created with all the necessary *Components* to provide a seamless integration with the framework. Some *Scenes* will also be provided with *"kitchen sink"* style examples of how to use the library and its components. The proposed asset package should use the Unity Oculus Integration (Oculus, 2024) as an example to follow on how to properly build a complex package for Unity projects.

The library will include a *singleton* `MonoBehaviour` manager class. This manager will interact with the `mediator` and handle all connections to `brokers` and relay of *messages* to the appropriate `personality` event handler methods.

A `BaseNodeInteractor` `MonoBehaviour` will be provided which will contain all the base logic needed to create a digital twin of a `node` on the network and expose any declared `personalities` to be handled in its internal events. The idea of this class is to be extended by the application in order to represent individual `node` configurations used by the application itself, thus allowing for a great deal of flexibility without compromising the implementation with predefined `node` types.

Each `personality` will have an associated `AbstractPersonality` abstract class that will provide:

- The associated *topic* related to the `personality` relative to the `node`'s base topic.
- An associated `UnityEvent` to provide a standardized interface with other components in the game engine.
- A method for registering the *topic* with the *manager* object for *topic* subscription and relaying purposes.
- A *callback* method that will be called by the *manager* whenever an associated *topic* message is received. This method will be responsible for decoding the message and firing any associated `UnityEvent`s.

A *node interactor* will consist of a collection of `AbstractPersonality` objects being managed by this class instance. This object will register any necessary event handlers for

all involved sub-objects and handle the pairing workflow and related connection events for the application.

The entire framework implemented in Unity will be event-driven using `UnityEvent`s whenever possible, guaranteeing the compatibility with existing projects and ensuring that the application is never blocked or has to pool the system in order to provide functionality. This `UnityEvent` based approach ensures that projects can have a plug-and-play approach to using `nodes` in the system, allowing for testing using mock devices in software, mimicking events, or by providing alternative controls using more standardized peripherals.

This event based approach also means that developers using the proposed library must be familiar with Unity's event system and must be comfortable with event-driven development.

With the ethos of using small base classes that are extended by the developer, borrowing from other integrations such as the one provided by Meta, the sustainability and flexibility of the proposed system is ensured, allowing for a great deal of customization in order to accommodate almost any scenario, while only incurring a steep learning curve at the beginning, and an upfront cost of development in order to extend the proposed base classes in order to represent the physical devices.

4.4 Embedded library

In the same vein as with the game engine library, the proposed goal is to provide a complete solution to creative application developers looking for integrating such new and exciting devices into their projects. With this in mind a library for the most common IoT development platform currently in the market, Arduino, will be developed. (Coherent Market Insights, 2024)

The library should be compatible with most IoT development boards that support development through the Arduino IDE, but most importantly it'll be certified to work with Espressif's ESP32 and ESP8266 devices, which are the most commonly used for IoT prototyping. (Espressif, 2023b)

Because of the tight latency and performance targets that an input device to an application such as a game or interactive experience may require, usually in the realm of less than 50 ms, (Long & Gutwin, 2019) the code for the firmware on the embedded devices should be kept to a minimum and the proposed library should account for this.

The proposed library will only expose an interface that allows for setup and a communication event loop, in line with the ethos of the Arduino platform. The library should also take advantage of hardware interrupts in order to ensure that general code written with the framework for the IoT module is kept to a bare minimum, offloading any further processing tasks to the much more powerful `client` running the experience itself.

The library will contain a class `Node` which will expose methods to be overwritten by the IoT developer to hook into internal events fired by the library, such as waiting for a connection to the `mediator`, entering pairing mode, finished pairing with a `client`, and an event for when messages arrive from the `client`.

Each device should use the `Node` class in order to provide an ID string, set by the developer upon the initialization of the object, list of `personalities` provided by the module, and

the device UID, which in most cases will consist of the device's Medium Access Control (MAC) address and calculated internally by the library, with no input from the developer. All of these parameters will be used by the `mediator` in order learn about the `node` itself and to pair it to a `client`.

It's important to keep abstractions in the embedded library down to a minimum due to the small processing power of the devices involved and the constant task switching from the Real Time Operating System (RTOS) found in most of these development solutions, (Espressif, 2023a) therefore the library will only abstract the initial connection, reconnection, pairing, and any non-latency sensitive parts of the process, ensuring that time critical parts are implemented by the developer, keeping the these sensitive code sections as tight as possible.

4.5 Mediation server: The missing piece

It's important to note that due to the inherit nature of the MQTT protocol there's no facility provided for MQTT clients to negotiate topics, these have to be either, predefined or hardcoded in both the publishing and subscribing client, or an external service must be used to provide a sort of pairing operation that links publishing and subscribing clients together and negotiates a common topic for them to use.

Commonly in many MQTT based systems, and specially in home automation systems with IoT devices using MQTT, a hardcoded address to a backend HTTP server with a REST API is used to connect the device for the first time and set up the connection to the *broker* and topics used from there onwards, along with any additional authentication and pairing that may also be required by the application in question. (Kim et al., 2015)

Since using another protocol, HTTP with a REST API in this case, is such an industry-standard mechanism of exchanging setup parameters with an MQTT *client* and provides a great deal with flexibility, the proposed framework will utilize a similar system in order to match and setup `nodes` with `clients`.

4.5.1 Technology stack

Anyone that's familiar with the technology stack of most modern web applications will feel comfortable deploying and managing a system that uses the proposed framework, as exemplified by Figure 18. The choices made are by design and intended to provide low friction to existing developers of creative applications and even enable multiple parts of the system running on the same physical machine if needed.

The server software will be written using the Python programming language (Python, 2024) due to its ubiquity in server-side and client-side applications, being the most widely used programming language since 2021. (TIOBE, 2024) In the cases where the `mediator` has to talk to `nodes` or `clients` over MQTT it'll do so using the `paho-mqtt` (Eclipse Foundation, 2024) library since it's considered one of the most complete and tested implementations of the protocol for the language. In order to provide the required REST API for `clients` and `nodes` to initiate pairing operations, the Flask web application framework (Pallets, 2024) will be used given its proven track record, being the *de facto* standard lightweight framework for building small REST APIs.

The recommended setup shown in the diagram will surely represent most of the deploy-

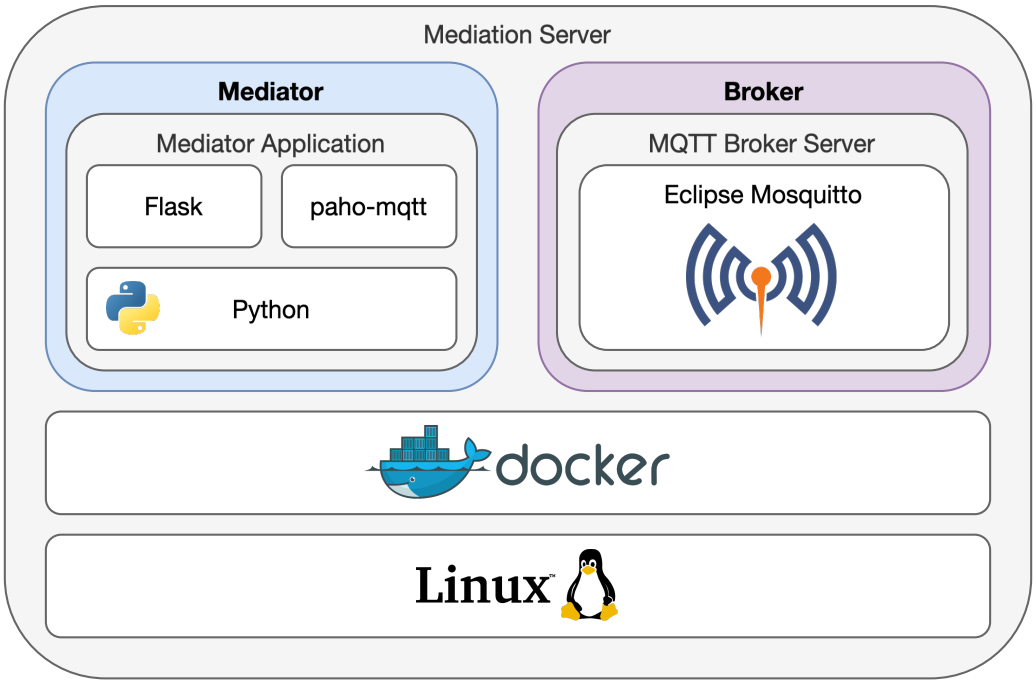

Figure 18: Diagram of the software stack used by the mediation server.

ments of the system. The `broker` and `mediator` will run inside containers on top of Docker on Linux, an industry-wide best practice and production-ready architecture for server software. The `mediator` software will be developed and tested only on Linux, although it may work on other platforms that provide the ability to run Docker and Linux based images, it wont be tested in such situations or be certified to work.

Due to the containerized nature of these pieces of software, their images and configuration would be streamlined and ensures that a system that works will continue to work for the forseeable future without receiving updates. Also, due to the isolation from the base system brought on by the containerization technology, the server software can be configured to run alongside the `application` in a powerful `client` capable of handling the workload.

Having the `application`, `broker`, and `mediator` all running on a single machine can bring some latency, cost, and complexity advantages, but must only be used in situations where the `application` won't benefit from the scalability of the distributed nature of separating each individual part of the system, and combining them into a single `client` is advantageous from a viability standpoint.

4.5.2 Discoverability of the server

In order to ensure that `nodes` on a system don't require the server's address to be hard-coded into their firmware, something that is prone to issues down the line when inevitably some part of the infrastructure gets updated, or if the server is part of a network without a static IP address assigned, a discoverability system was devised.

`Nodes` may discover the address of the server by sending a UDP broadcast message to the `mediator`'s default communication port, which in turn, if running, should reply with the appropriate IP address to communicate with and perform any queued pairing operations,

working in a fashion similar to the way DHCP works. (Perkins & Jagannadh, 1995)

After a `node` receives the `mediator`'s IP address it should save it internally to be reused on the next power up cycle. If after powering up it's unable to communicate with the cached `mediator`, only then it should attempt discovering a new one using the broadcast method. This avoids a self-inflicted DoS on the network if a large number of `nodes` try to broadcast their discovery packet. (Nuiaa et al., 2021)

With the discoverability of the server dealt with, the only information that has either to be entered during setup or hardcoded into the `node`'s firmware is the network it should connect to, everything from that point forward is fully handled automatically by the framework.

4.5.3 Solving the pairing issue

Pairing resource constrained devices such as IoT modules that usually don't offer much of an user interface has always been a problem for the industry. Usually such devices are either pre-configured with hardcoded values or require the usage of another device to be complete their initial setup. (Bahga & Madisetti, 2014)

Apart from the network setup, due to the nature of the MQTT protocol, the `node` will also require an extra step in order to setup itself with the `broker` and the correct `client`. This is where the proposed framework will solve one of the most egregious problems in systems such as these: Pairing similar devices to the correct `client`.

4.5.4 Connecting to the network

The most widely adopted solution is that upon power on a device that's not configured to connect to a wireless network will start as an access point that another device must connect to, where an embedded web page or REST API will be used to configure the IoT module and connect it to another access point, after this point the configuration is saved and that initial mode can only be entered again in case the cached network has to be changed.

The proposed framework will use the same industry standard approach to the initial setup of the `nodes`. A developer will have the ability to hardcode the desired wireless network in the `node`'s firmware, although it will also provide the possibility of entering the setup mode where a new network may be selected. For security and integrity reasons the developer will be able to restrict access to the setup via a password. If the setup is not finished, upon the next power up of the module it will use the last saved parameters as a fallback mechanism.

4.5.5 Pairing a node to a client

After the initial setup of the `node` on the network is done, the breakthrough in this research is on the mechanism to pair and exchange information between `nodes` and `clients` without having pre-exchanged any information beforehand.

The proposal, as shown in Figure 19 is a system that when looked at from the eyes of the user will work in a similar fashion to the way wireless game controllers are paired. The `mediator` will be waiting for incoming pairing requests triggered by `clients`. When a suitable `node` powers up and announces itself the `mediator` will immediately pair it to

the client that requested a node with matching characteristics.

Upon starting an application, it will contact the mediator via its REST API and initiate a pairing request by registering itself with its Unique Identifier (UID), based on client parameters such as its MAC address, and its required node IDs and their respective personalities. The mediator will reply with a *pairing UID* for each requested node. This will guarantee that nodes with similar personalities that have different usage won't be confused by the mediator while the pairing operation is underway. The application will wait for pairing updates from the mediator by *subscribing* to an MQTT *topic* with the following namescheme: <application uid>/pair/<pairing uid>.

After the pairing request is initiated by the application, the mediator will wait for nodes with matching UIDs and personalities to power up in pairing mode and request a client to pair to. Whenever this happens, and matching client is found, the node will be notified and should start *publishing* its data to MQTT *topics* with addresses following the convention: <node id>/<node uid>/<personality>.

When the pairing of a single node to an application is considered to be viable from the eyes of the mediator, it will notify the client via its *pairing topic* of the viable node's UID, which in turn will start subscribing to all the required *topics* related to its personalities. After the entire process is done, the application will notify the mediator via its REST API that the pairing for that node has finished using its *pairing UID*. The mediator will then consider that pairing operation as completed and will drop it from the list of pairing operations in progress.

At this both, the node and client, have been paired together and can cache their respective MQTT *topics* for use on next power up, dismissing the requirement of pairing devices each time they power up. This is important since in most installations the setup will be done once and shouldn't be changed later, only if needed to update it or replace parts.

It's important to note that all pairing operations using MQTT must happen with a QoS level of 2, ensuring that all parties involved receive the important pairing messages.

By basing the MQTT *topic* schemes on the node's Unique Identifier it's ensured that there won't be a conflicting situation where multiple nodes with cached client *topics* could be controlling the same client. Switching the *topic* namespace to the node may only create a rare situation where multiple clients may be subscribing to *topics* from the same node, causing it to control two applications at once, although this situation can easily be mitigated by ensuring clients always are required to ask the mediator if its cached list of nodes aren't being used by another application, and if so they should be forced to invalidate their cache and register another pairing request for the offending node UIDs.

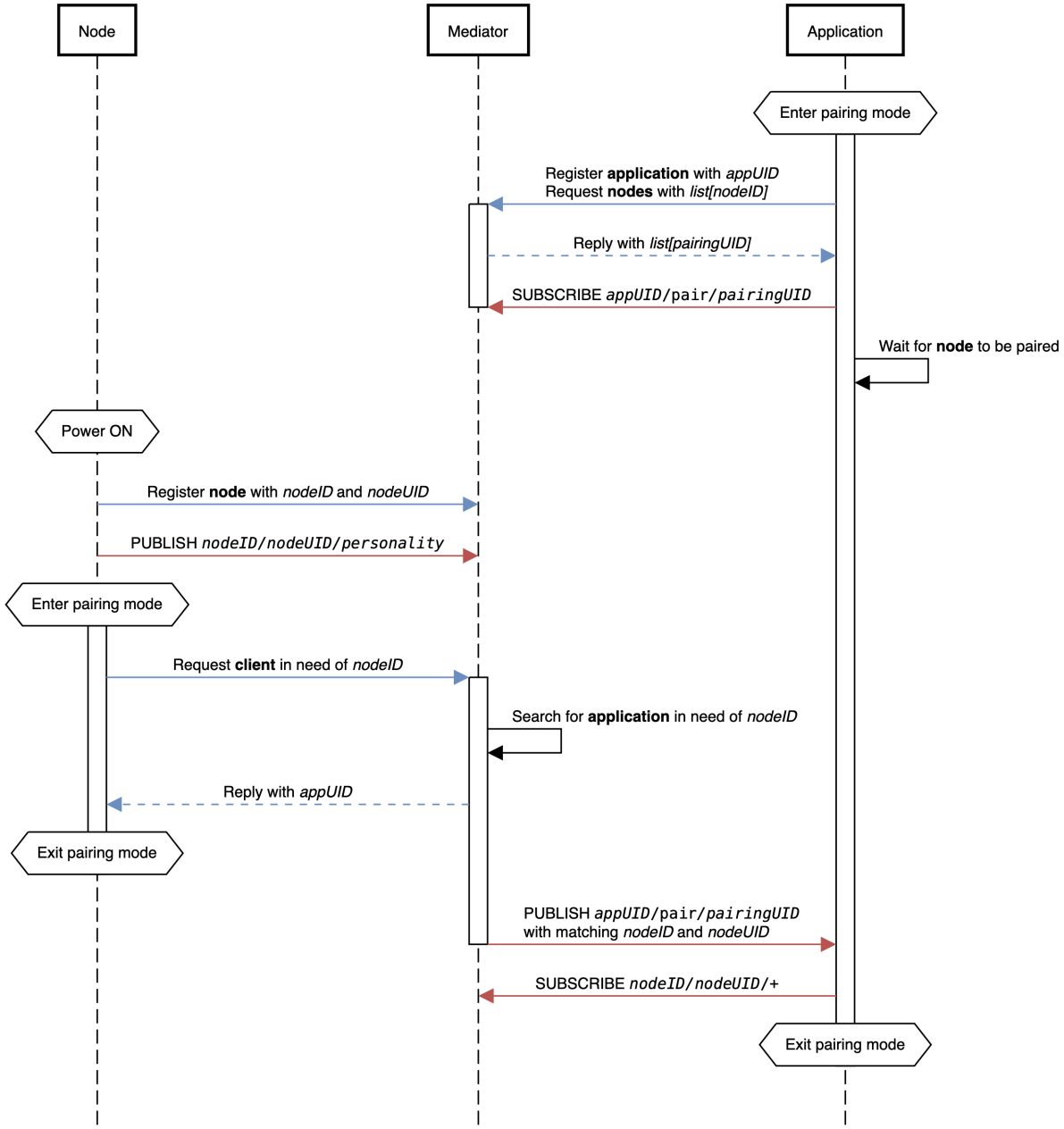

Figure 19: Sequence diagram of the pairing workflow.

4.6 Opening new possibilities

With all the solutions proposed by the framework, bridging the gaps involved with integrating IoT devices into creative applications with the same ease of setup as wireless controllers is clearly an attainable goal. This can open up the applications targeted by this work to an entire new world of possibilities, allowing for truly immersive experiences to be produced at a room scale, no longer being bound to the confines of a screen or the extend afforded by PC peripherals.

Creative applications such as the ones exemplified throughout the conducted research are the main objective to solve for in this system, although some of the solutions discussed can surely benefit many other areas where MQTT is used.

General industrial and home automation applications can apply the same concepts discussed here and create applications that easily integrate new devices into their respective systems bypassing much of the tedious, and some times error prone pairing systems currently in use.

4.7 The viability breaking point

So far the proposed framework has lots of potential and is capable of solving multiple problems commonly found when implementing such integrations between IoT devices and creative applications, but due to its architecture the applicability of the framework leans heavily on the fact that the MQTT protocol is performant enough to provide acceptable performance for the intended applications.

Given the fact that the protocol is incapable of communicating directly with `clients` and must go through a `broker`, adding another point of interaction where latency may be induced, it's crucial that the viability of the protocol for the proposed application in terms of low latency are actually fulfilled.

The purpose of the next section is specifically dedicated to testing if leaning the entire system, that requires low latency, on a protocol that requires a `broker` is viable and acceptable performance is achievable, thus proving that the entire structure of the framework being proposed is sound.

5 MQTT protocol performance evaluation

The proposed system is heavily based on network communications using MQTT, and due to constraints imposed by its main use case in creative applications such as immersive experiences and games, the communication must be efficient and provide an acceptable level of performance to the user.

The system's intended use in creative applications as a means of integrating IoT devices into game engines, means these devices will be used as input devices, controllers, and in some cases output devices. Some of the integrations may require the latency of the communication to be at a minimum, such as a device that's working as a controller and must provide an acceptable experience to a player, some may have a lot of tolerance to delays on the network, if for example an application wants to read the temperature from a sensor, it'll be doing so every second or half-second, neglecting any influence of network latency.

Another important factor is the fact that the proposed system must account for is the bandwidth capabilities of the network and how much overhead it may be incurring for every transaction in the system. Even though MQTT is a very efficient protocol (Naik, 2017) the system must account for the scenario where multiple devices using the framework may be connected to a congested network, as is usually the case for immersive experiences being showcased at public events, and thus it's important to profile the impact of having constant communication with a relatively large number of devices using the system in order to predict the impact that it might have on a network and in the perceived responsiveness of implemented applications.

Other system parameters such as the hardware resource utilization of the server are an important factor to account for when dimensioning a machine and should also be profiled in order to estimate the requirements of a server acting as the broker in the system. (Sun et al., 2016)

In order to get a broader picture of how the MQTT protocol behaves when utilized in a distributed and constantly publishing scenario such as the proposed system, a series of experiments were conducted in order to profile its behavior and determine if the protocol is actually suited for the tasks required by the use case.

5.1 Methodology

Unless otherwise noted, in order to achieve consistency and repeatability in the results, all the tests in this chapter will be performed using two identical VMs running on the VMWare Fusion™13 virtualization platform. All of the network communication will be accomplished between these two systems in a private virtualized network provided by the same virtualization software that's hosting the machines.

The configuration of the two identical virtual machines used are as follows:

Operating System Debian Linux 12 (Kernel version 6.1.0)

Num. Processor Cores 4 (shared)

Memory 2048 MB

Architecture x86-64

Network Interface Intel®PRO/1000 (virtualized)

The computer hosting these virtual machines and running all the software necessary to support this project is an Apple MacBook Pro 16-inch (2019) (EveryMac, 2019) with the following specifications:

Operating System macOS 14.4.1 (23E224)

Model Identifier MacBookPro16,1 (A2141)

CPU Intel Core i9 2.4 GHz (I9-9980HK)

Memory 32 GB (DDR4 2667 MHz)

Architecture x86-64

It must be disclosed that the host computer of the virtual machines was not dedicated to the task of running the tests, being a general purpose computer, and had other applications running simultaneously during the tests which may have slightly impacted the gathered results, although given the specifications of the machine and the fact that unnecessary software was closed before running any critical tests, it was assumed that the effect of other applications was negligible.

In order to easily distinguish between the test machines the host computer will be co-denamed `earth` and the two identical virtual machines involved in the tests were named `uranus` and `saturn` respectively and will be referenced as such moving forward whenever a distinction between the two must be drawn.

5.2 Baseline

In order to provide a basic baseline of the network capabilities of both systems some industry standard tests were performed to measure the performance of Internet Control Message Protocol (ICMP) and TCP traffic (Blum, 2003) between both virtual machines and between the virtual machines and the host system. These simple tests using standardized tools provide a baseline of performance for the best case scenario of communication between the systems without the overhead of the proposed software or the MQTT protocol itself. It's important to emphasize that these tests are intended to stablish a baseline and do not represent real-world scenarios.

Since the `uranus` virtual machine is the one that will be hosting the MQTT broker software in later tests all of the tests performed in this section will be done from one of the test machines to the `uranus` VM.

5.2.1 Benchmarking ICMP performance

The first tool to run whenever there's a need to test the network between two machines is to use the `ping` command. `ping` is a widely available, small utility, that allows administrators to query the availability of a network and determine if two machines are actually capable of talking to each other. It uses the ICMP protocol to communicate and measure the Round Trip Time (RTT) of the associated network, and thus can be used to determine the latency of the overall connection. (Blum, 2003)

Machine	Min.	Mean	Max.	Std. Dev.
`saturn`	0.308 ms	0.568 ms	1.050 ms	0.152 ms
Host	0.240 ms	0.416 ms	2.186 ms	0.210 ms

Table 1: `ping` test from host and `saturn` to `uranus`

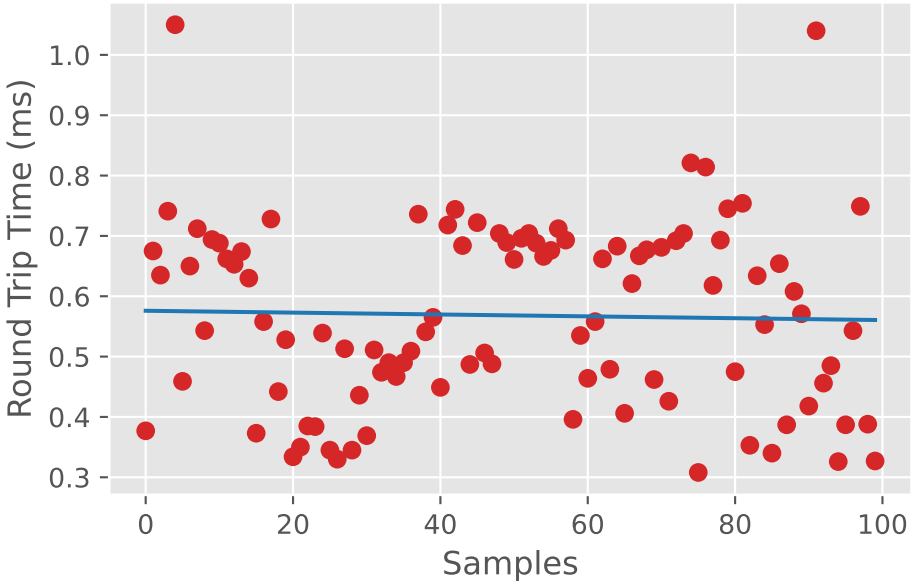

Figure 20: Round trip time of 100 `ping` samples from `saturn` VM.

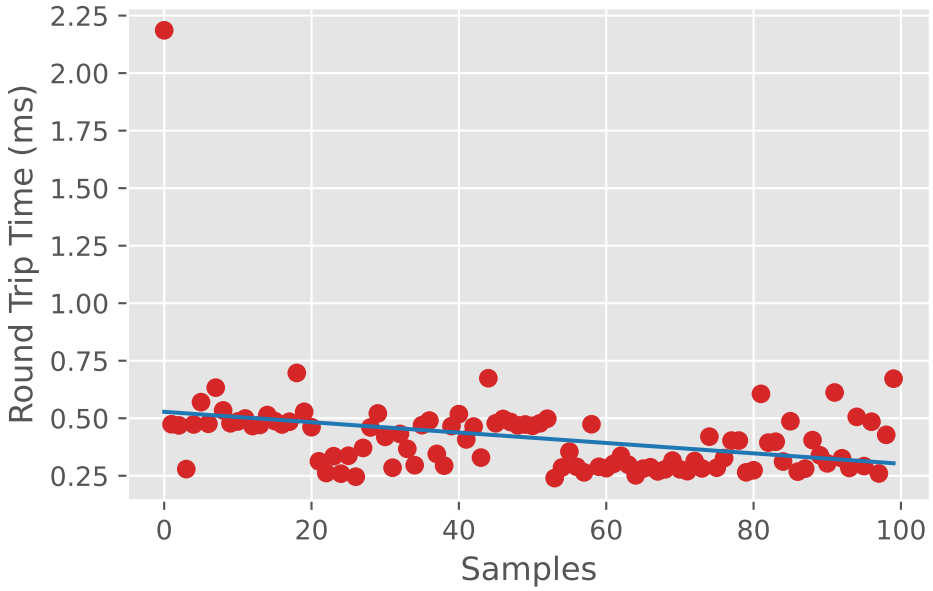

Figure 21: Round trip time of 100 `ping` samples from host machine.

From the test data acquired in Table 1 and figures 20 and 21 it's possible to infer a base average round trip latency of 0.568 milliseconds from `saturn` and 416 milliseconds from the

host machine when using the ICMP protocol for benchmarking with `ping`, demonstrating that there are no limitations and that the network operation is running as quickly as possible given the speed of the machine's processor itself and not slowed down by any additional infrastructure.

5.2.2 Benchmarking TCP performance

Although the results from the `ping` command are widely used and accepted by network administrators as a valid measurement, its use of ICMP doesn't reflect real world applications, which generally use TCP, such as HTTP and MQTT, both of which are the basis of the proposed framework. This is a crucial distinction because some networking gear may prioritize TCP traffic. (Phanekham & Jones, 2020)

In order to get a better sense of how the TCP protocol performs in this test lab, another application must be used, in this case `netperf`. This industry standard tool uses TCP to test the network performance and is capable of performing proper latency and throughput experiments. (Blum, 2003)

Since `netperf` uses TCP for its test and is extremely flexible and configurable, it's possible to configure it in such a way to perform similar to how MQTT would, specially in terms of the length of each TCP packet, allowing for a much closer comparison to how a system would behave in reality. Given the fact that a `PUBLISH` packet for an IoT sensor sending a small quantity of data it has collected, usually a single data point such as a position, angle or temperature value for example, is usually on average around 32 bytes for the entire MQTT packet, a test using `netperf` was setup to send this length of data to measure the RTT and throughput.[1] The results of the tests can be found in tables 2 and 3.

Latency				**Transaction Rate**	**Throughput**
Min.	Mean	Max.	Std. Dev.		
0.153 ms	0.556 ms	80.65 ms	0.900 ms	1796 T/s	1795.99

Table 2: `netperf` test results for `saturn` virtual machine

Latency				**Transaction Rate**	**Throughput**
Min.	Mean	Max.	Std. Dev.		
0.072 ms	0.385 ms	422.18 ms	1.091 ms	2591 T/s	2591.96

Table 3: `netperf` test results for host machine

The `netperf` test concluded with a base average round trip latency of 0.556 milliseconds from `saturn` and 0.385 from the host machine when using the TCP protocol, very close to the 0.568 milliseconds figure from the ICMP test using `ping`.

[1] The command used for the TCP test using the `netperf` utility:

```
netperf -H uranus.thesis -l 100 -t TCP_RR -v 2 -- -o min_latency,
    mean_latency,max_latency,stddev_latency, transaction_rate,
    throughput -r 32,32
```

By analyzing the data collected so far it's safe to assume that subsequent tests of the MQTT protocol itself won't be skewed by the underlying infrastructure and can be trusted to be an actual test of the protocol itself without any limitations imposed by the test suite hardware.

5.3 Benchmarking MQTT

After the standardized set of tests were run in order to determine that the hardware used in the tests was not a limitation, the next set of tests were aimed at benchmarking the actual performance of the MQTT protocol itself and all of its required software stack.

Commonly, a project that uses MQTT for communication has a couple of separate pieces of software that are developed and maintained separately, this is largely due to the fact that the protocol requires a *broker* for *clients* to communicate with. Given the role of the *broker*, relaying messages back-and-forth between *publishers* and *subscribers*, this piece of software is rarely embedded into a server application (Espinosa-Aranda et al., 2015) and is generally available as a standalone server application that other pieces of software will connect to, similar to how databases and reverse proxies are usually deployed. (Belli et al., 2015)

Due to the distributed nature of the deployments of this technology the only piece of software that may be tested for its impact in the performance of any system that utilizes MQTT is the broker software, since this part is also common to many different deployments and software stacks, this implies that the results may also be reflected on other systems that also utilize a similar software stack as the one tested here.

Just like any other software that's used as the backbone of a protocol, there are many broker implementations from many different vendors available, some are specifically designed to be used in specific scenarios such as distributed systems (Longo et al., 2020), embedded (Espinosa-Aranda et al., 2015), or based on peer-to-peer technology (Buccafurri et al., 2023), and some are designed to be more broadly deployed and used as a general purpose broker for most applications.

Since the proposed solution is largely to be deployed on-premise or on a small cloud computing instance to serve the needs an interactive or immersive experience this section and the proposed framework itself will use a general purpose broker software.

Due to the requirements imposed by the proposed use case in creative applications it is of utmost importance that the broker software chosen for the proposed system be capable of handling many connections at once and being able to accept and relay them quick enough in order to not cause noticeable latency issues for users (McCann & Pollard, 2007) when MQTT-connected devices are used as controllers or inputs in games or immersive experiences.

In order to test the best case scenario in terms of communication between *clients* and the *broker* in the proposed system a bare-minimum version of the *publisher* and *subscriber* code of the proposed software framework was developed. In order words, a thin implementation over existing MQTT libraries for the technologies involved in order to profile only the impact of the broker software and the communication libraries that will be imported in order to implement the framework. (Beyer et al., 2019) Creating the minimum viable

source code is important since it'll remove any variables and uncertainties introduced by "business logic" of an application that implements the framework or by the logic of the framework itself and the overhead that high-level software abstractions create in order to expose a developer-friendly API.

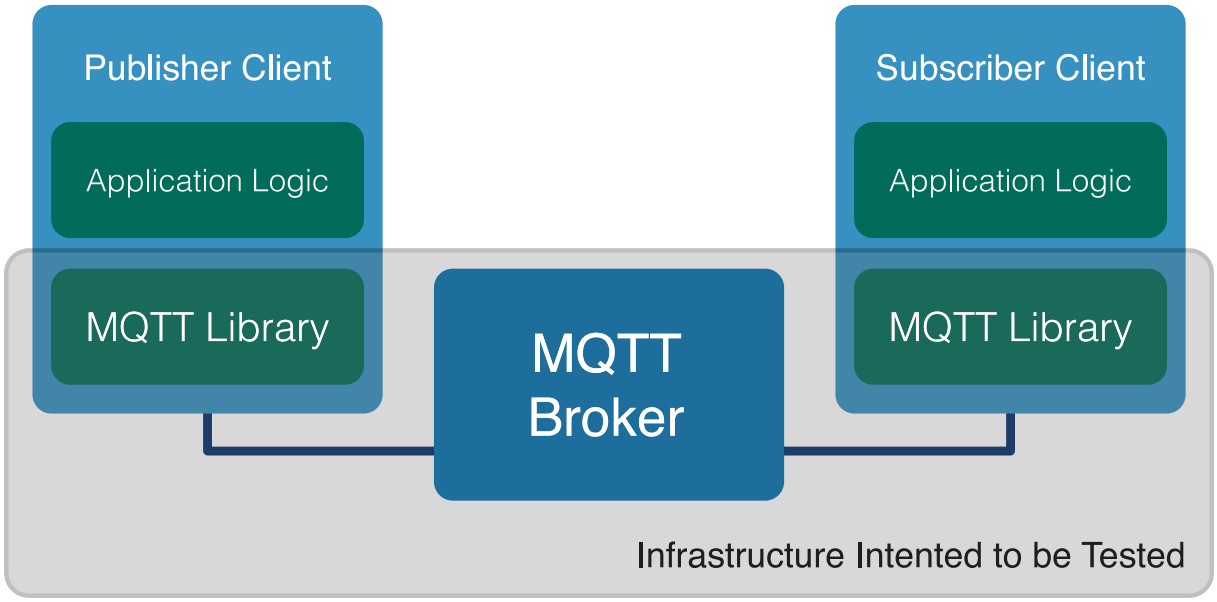

Figure 22: Grey area representing the software stack intended to be benchmarked.

5.3.1 Choosing a broker

The MQTT broker software is the most important piece of software in the framework's stack since it's the central hub of communication among all of the devices in the proposal. Given its importance to the system it's important that the software chosen must meet and exceed all of the requirements in terms of latency and throughput in order to give the largest amount of breathing room possible for inefficiencies on client's libraries and also for the logic and implementation of the user's applications.

Mishra et al. in their article "Stress-Testing MQTT Brokers" (Mishra et al., 2021) provided valuable insights into the performance characteristics of many popular and widely deployed broker server applications. In their article they analyze in depth the load performance of Eclipse Mosquitto (Eclipse Foundation, 2018), Bevywise MQTT Broker (Bevywise, n.d.), ActiveMQ (Apache Software Foundation, 2024), HiveMQ CE (HiveMQ, 2019, April 1/2024), VerneMQ (Octavo Labs, n.d.) and EMQ X (EMQ Technologies, n.d.), which combined represent the vast majority of broker solutions available on the market currently, due to this fact the choice of broker must come from this list. Some of their work can be seen in Table 4 and Figure 23.

In order to ensure the longevity and maintainability of the solution, given the environment in which it will most likely be deployed, which requires a very long service life, the Bevywise broker was rejected due to being proprietary, and without the source code being available it's not certain that the project can be maintained in case its governing entity decides to cancel or abandon the product.

Brokers	Average Latency in ms.		
	QoS0	QoS1	QoS2
Mosquitto 1.4.15	1.65	0.74	1.38
Bevywise MQTT Route 2.0	1.13	0.96	1.53
ActiveMQ 5.15.8	2.33	1.38	2.14
HiveMQ CE 2020.2	7.69	58.48	3.66
VerneMQ 1.10.2	1.53	1.98	2.89
EMQ X 4.0.8	1.34	0.87	3.68

Table 4: Latency comparison of local brokers. (Mishra et al., 2021)

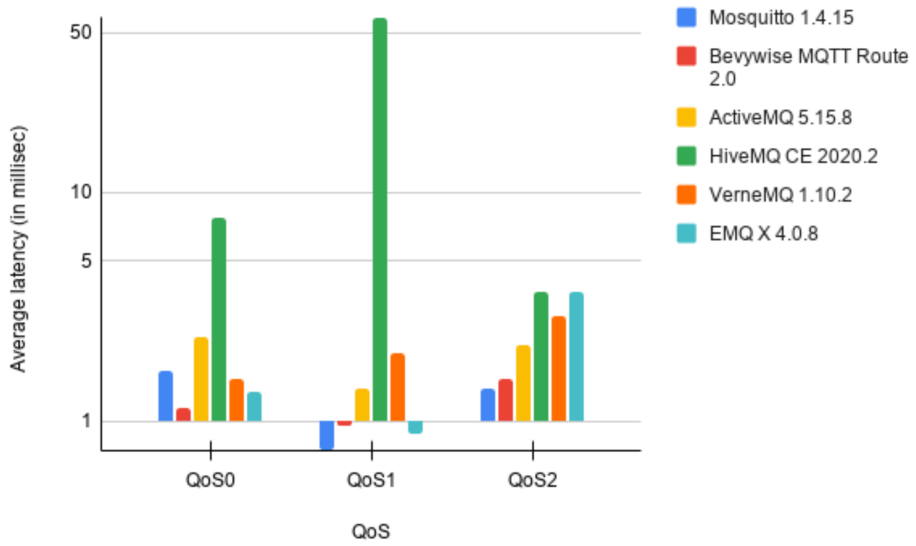

Figure 23: Comparing the impact of different QoS levels. (Mishra et al., 2021)

Upon analyzing the findings from Mishra et al. it was safe to assume that clearly the most performant broker server tested was Mosquitto, a server that's well regarded as one of the industry's *de facto* standard brokers. This server apart from leading in performance and active deployments it's considered a reference implementation, meaning it will most likely get new features in future versions of the protocol before all others. This coupled with its open source ethos and proven track record, compels it to be the best choice for the application.

5.3.2 Choosing the client libraries

As previously discussed the goal here is to benchmark the lower level of the system's architecture, providing only the bare minimum needed for communication between two MQTT clients using standard libraries in order to implement the protocol. Given the fact that the broker software for the tests has already been chosen the next step is to select the libraries to implement the clients with.

The test will simulate a real world scenario of a *publisher client* communicating a lot of real-time data with a *subscriber client*, this will be a common scenario in the use case of the framework since it'll most likely be used to continuously gather information about

the environment an immersive experience takes place or gather information from another custom input device that requires a high refresh rate.

To simulate the *publisher client* a Python script was used since it's not only a widely used language due to its ease of use, platform support, and plenty of libraries available, but also because it's very likely that solutions built in this creative field may likely use Python instead of a compiled systems language such as C++. The most widely used MQTT library for Python currently is `paho-mqtt` (Eclipse Foundation, 2024), which is developed by the Eclipse Foundation and is an official wrapper of the C library used by Mosquitto to implement the protocol so it's guaranteed to be performant and future-proof given the success of its broker counterpart.

In order to simulate the *subscriber client* the Unity game engine (Unity Technologies, n.d.) was chosen as the test bed of an creative application, since many interactive experiences and immersive applications are currently built using it, this is an accurate representation of a real world application environment. In order for the Unity application to communicate with the MQTT broker a third-party library is needed since the game engine doesn't have support for the protocol out-of-the-box, in this case the `M2MQTT for Unity` (Viganò, 2018, July 23/2024) library was chosen due to it also being based on Eclipse's Paho library with a thin wrapper for Unity and being endorsed in examples by important companies in the industry. (EMQX Team, 2023)

5.3.3 Methodology

As perviously explained, the test setup will consist of applications that are only thin wrappers around the chosen libraries in order to properly profile the characteristics of the system as a whole without any interference from the logic of the applications that will be implemented on top of the proposed framework. (Barker & Shenoy, 2010)

The *publisher client* will be a simple Python script, running in the host machine, separate from `uranus` that's running the Mosquitto broker, that will be constantly publishing updates to the test topic with the timestamp when the updated was sent as its payload as fast as it can. It will also test the broker and the subscribed clients by simulating being multiple *publisher clients* at once sending data to multiple topics, testing a system with many distributed devices.

The *subscriber client* will be a bare minimum Unity 3D application, running on the host machine, that will receive the message published by the Python script and calculate the delta between the message being sent by the publisher and it being received by the application, thus calculating the response time of the entire system from publisher to subscriber. The application will only contain an UI element to display the collected information for later analysis. This client will only be capable of simulating a single subscriber in the system, although, by its very nature, the use cases will usually be comprised of many publishing clients for very few subscribers.

It's important to note that before any of the tests the host machine and the guest virtual machine were synchronized with the same Network Time Protocol (NTP) server. Another important point was that during all tests the Unity application was run inside the Unity Editor. Although this decision will affect the benchmark negatively the tests should still represent the experience a developer will have while developing using the framework.

5.3.4 Profiling a single publishing client scenario

In this scenario the *publishing client* will try to send messages in an extremely quick succession as soon as the broker will allow. The idea here is to gauge the bare performance of a single client on network trying to send as much real-time data as is possible in a fire-and-forget type of transmission. (Barker & Shenoy, 2010)

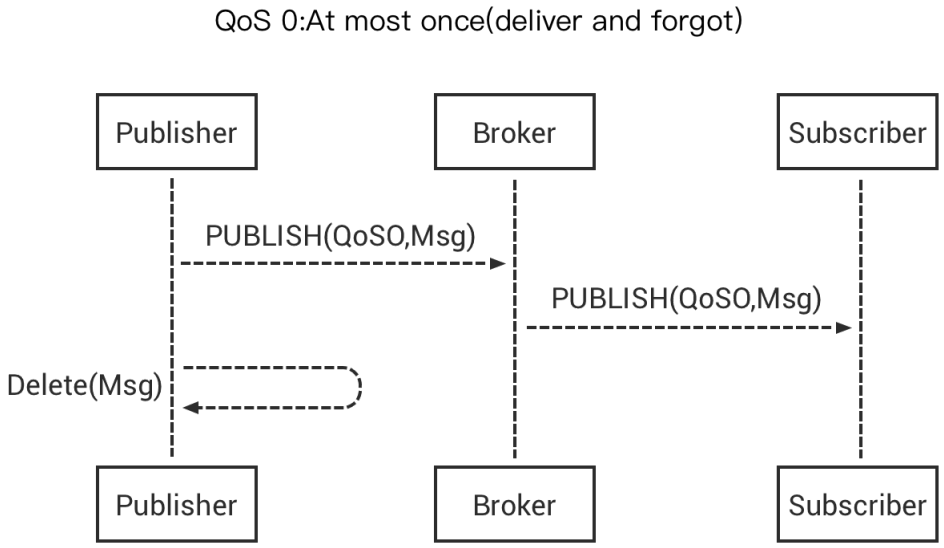

Figure 24: Example of an transaction using the MQTT's QoS level 0. (EMQX, 2020)

When this test was run for the first time Unity became unresponsive and had to be terminated forcefully. The simple act of receiving so many messages at once and trying to update an UI element so fast was enough to completely break the IDE. Although when the same test was run without making any changes to the UI the *Editor* was responsive the entire time and could keep up with the decoding of the messages perfectly fine.

In order to test the IDE's behavior further the test was run without updating the UI but instead logging the messages to the *Console*. The IDE was unresponsive while the messages were being received, but as soon as the test script stopped sending messages the *Editor* became responsive again. This behavior can be noticed in the data plotted in Figure 25.

Clearly, due to the intricacies of the internal logic in the game engine, updating elements at such a fast pace is not possible, even though computations can be made in the background, something the developer must be mindful of. It's important to note that while Unity was unresponsive the CPU usage was low, owning to the fact that the updates were happening in the main thread.

Considering the fact that Unity is unable to finish the test if updates are done to the UI at the rate needed for a proper benchmark all subsequent tests were done by writing the received values to a variable and saving it to a file after the test was concluded.

Delay	Min.	Mean	Max.	Std. Dev.	Loss	Throughput
0 ms	2.58 ms	1988 ms	3443 ms	995.54 ms	49057	35 436 M/s
1 ms	0.33 ms	1.59 ms	42.53 ms	3.82 ms	0	853 M/s
2 ms	0.36 ms	1.29 ms	42.48 ms	2.44 ms	0	442 M/s
3 ms	0.36 ms	1.29 ms	41.72 ms	2.30 ms	0	294 M/s
5 ms	0.43 ms	1.36 ms	38.51 ms	2.25 ms	0	172 M/s
10 ms	0.48 ms	1.38 ms	31.24 ms	1.89 ms	0	84 M/s
50 ms	0.58 ms	1.33 ms	7.89 ms	0.96 ms	0	18 M/s
100 ms	0.54 ms	1.12 ms	2.91 ms	0.46 ms	0	9 M/s

Table 5: Single publisher MQTT stress test results

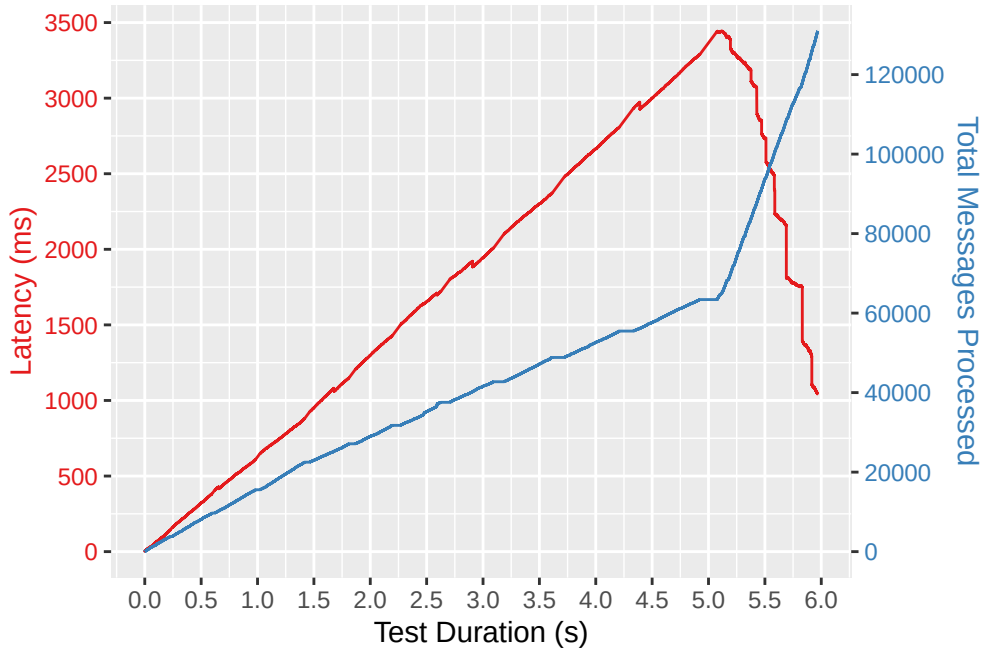

Figure 25: Latency and total processed messages over test duration for a single publisher without any delay between messages.

It's important to note that when Unity becomes overwhelmed with incoming messages its default behavior is to start dropping them and not continuing to process. More interesting yet is the behavior of the latency curve on the no delay test, Figure 25, where it has a very distinct linear slope until a point where it changes the angle of the slope completely.

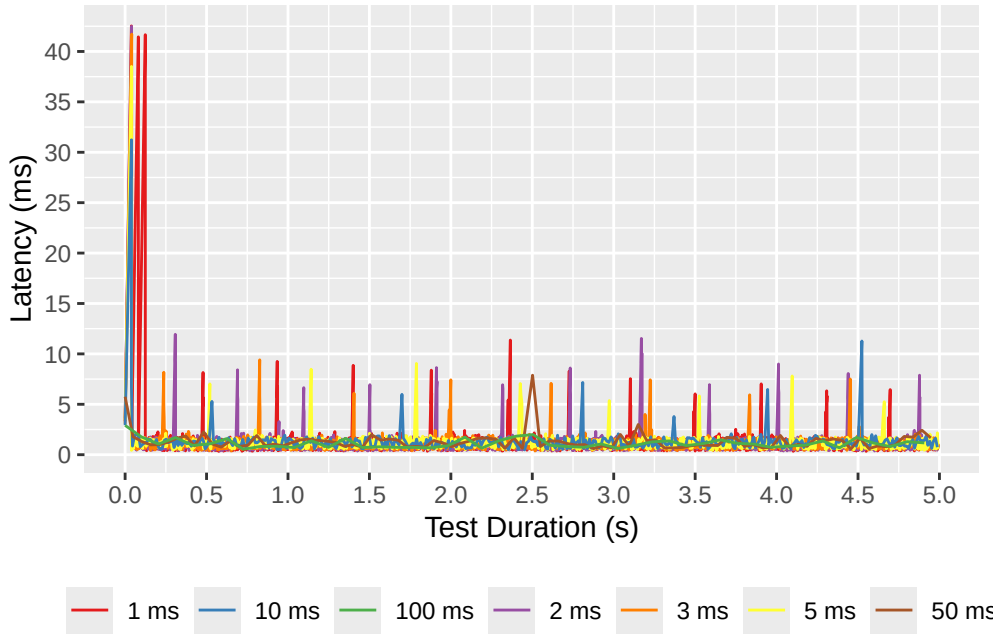

Figure 26: Latency over test duration for a single publisher with delays between messages.

With latencies measured in Figure 26 always hovering around 1 ms, and depending on what the engine is currently processing, single peaks of 10 ms being noticed, makes the latency caused by the framework itself negligible, giving headroom for any possible latency increases to the network infrastructure.

With these latency figures above it's safe to assume that, in these conditions, it's possible to use an MQTT device even as a game controller since the latency that's acceptable for such an input device is on average 50 ms (Jota et al., 2013), with the just noticeable difference determined to be around 10 ms (Ng et al., 2012). This opens up a lot for opportunities for new devices being user in new and creative applications.

For controller applications acknowledging the success of the data transaction is usually not needed since the priority is usually responsiveness, but some applications may require a relatively high degree of responsiveness while still retaining the ability to ensure the acknowledgement of the data exchanged.

An example where guaranteeing the arrival of the information is crucial to the application's success would be cases where the sent information is relative to its previous history such as the movements of an accelerometer. In order for the application to keep track of such a sensor it must be capable of receiving all of its readings with confidence and sequentially, otherwise it'll loose track of the physical object.

The MQTT protocol provides Quality of Service levels to ensure the delivery of published messages, allowing for applications such as described where there must be confidence that the information published arrived at the subscribed clients. (Zhou, 2023)

With QoS level 1 the *publisher* will wait for an acknowledgement from the *broker* immediately after it has published the message to any subscribed clients. It's of note that the acknowledgement at this level doesn't guarantee the delivery of the published message to subscribed clients, only that the message was sent.

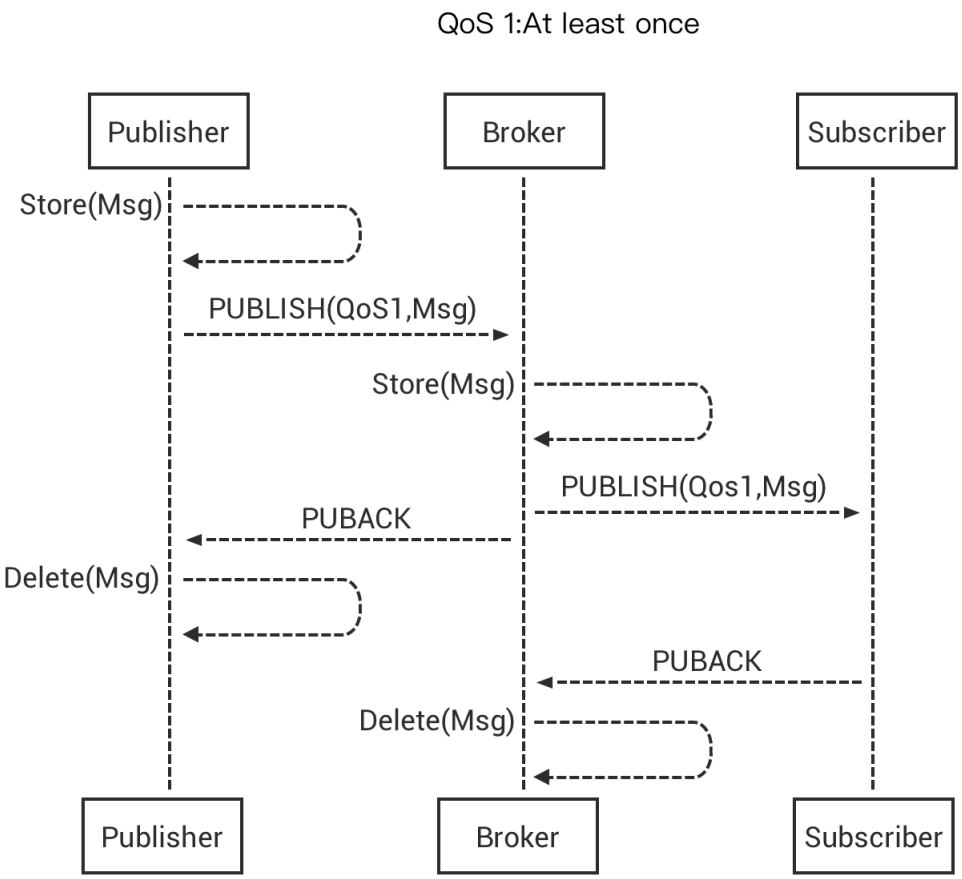

Figure 27: Example of an transaction using the MQTT's QoS level 1. (EMQX, 2020)

If the *publisher* requests a Quality of Service level of 2 it'll will wait for an acknowledgement from the *broker* after it has published the message to any subscribed clients and has received an acknowledgement back from at least one of them. This guarantees to the *publisher* that at least one of the *subscribers* has received the message successfully.

The same battery of tests was again performed with QoS levels 1 and 2 and the next message was only sent after and acknowledgement of the previous one was received. Any delays mentioned were applied after the acknowledgement of the previous message was received.

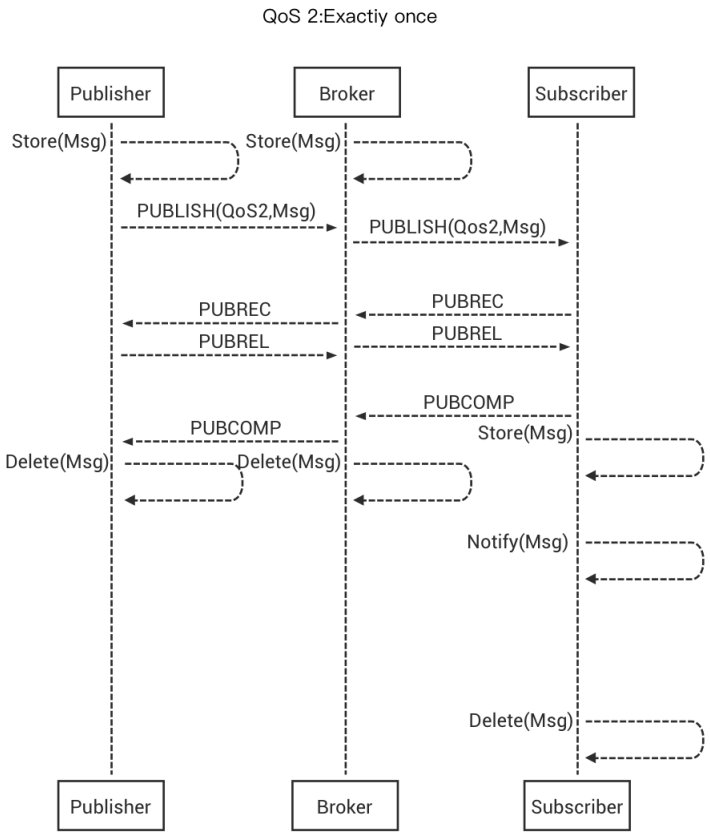

Figure 28: Example of an transaction using the MQTT's QoS level 2. (EMQX, 2020)

Delay	Min.	Mean	Max.	Std. Dev.	Loss	Throughput
0 ms	0.22 ms	1.04 ms	12.64 ms	0.82 ms	1	2693 M/s
1 ms	0.32 ms	1.12 ms	10.27 ms	0.87 ms	0	625 M/s
2 ms	0.30 ms	1.19 ms	17.53 ms	1.08 ms	0	365 M/s
3 ms	0.35 ms	1.28 ms	21.13 ms	1.25 ms	0	254 M/s
5 ms	0.40 ms	1.27 ms	11.71 ms	0.85 ms	0	156 M/s
10 ms	0.41 ms	1.30 ms	8.06 ms	0.70 ms	0	79 M/s
50 ms	0.58 ms	1.33 ms	4.97 ms	0.67 ms	0	18 M/s
100 ms	0.53 ms	1.35 ms	6.41 ms	0.95 ms	0	9 M/s

Table 6: Single publisher MQTT stress test results (QoS 1)

Of note in Table 6 is the fact that as soon as a QoS level that requires an acknowledgement is instated the game engine is no longer overloaded and can easily cope with the load, since it's no longer a barrage of messages due to the inherit delay introduced by the broker acknowledgement. Still with no delay applied Unity consistently dropped a single message on every single run of the test.

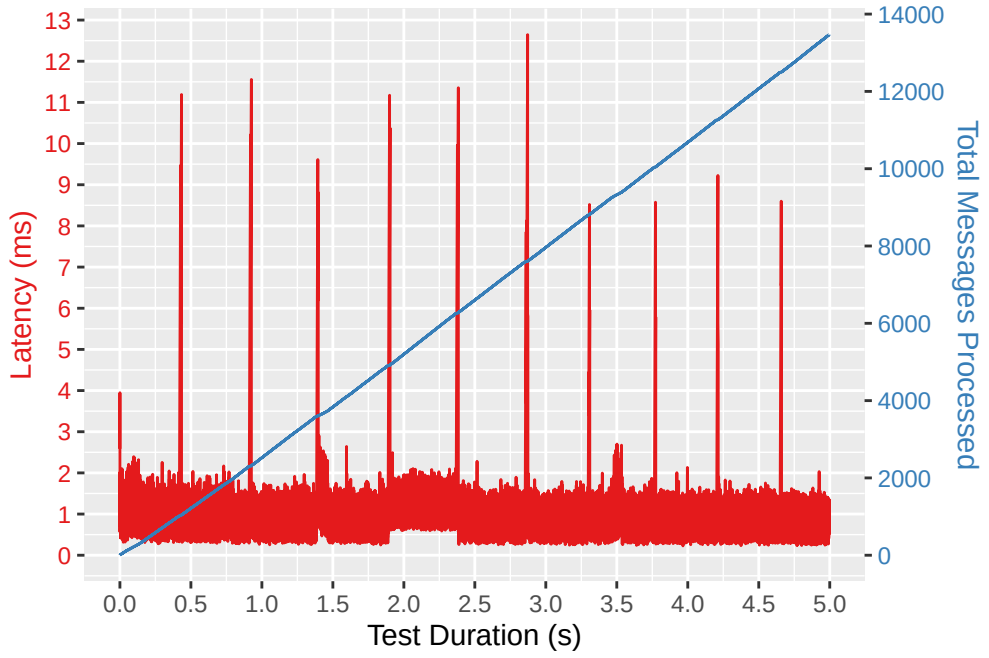

Figure 29: Latency and total processed messages over test duration for a single publisher without any delay between messages with a QoS level of 1.

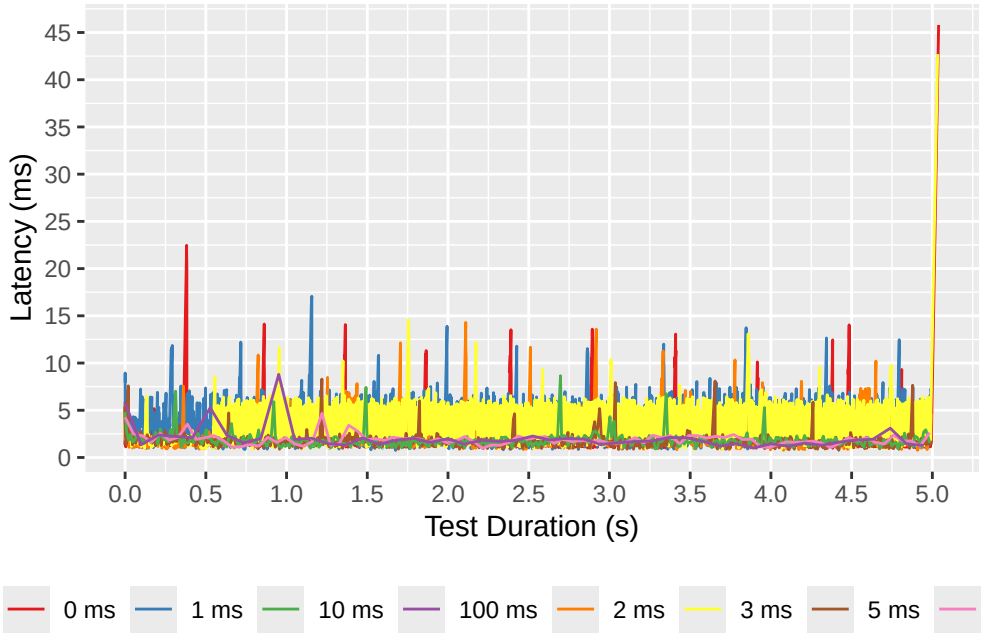

Figure 30: Latency over test duration for a single publisher with delays between messages and a QoS level of 1.

With a Quality of Service level of 1 there are some assurances related to the delivery of messages, although not perfect, at least in the tests shown in Table 6, almost all of the messages arrived at their destination, although it's important to note that this is a controlled environment, but it's still promising. Even with the overhead of the ac-

knowledgement it was still possible to achieve quite a substantial throughput and average latency figures well below the required for an user to notice. (Long & Gutwin, 2019)

Delay	Min.	Mean	Max.	Std. Dev.	Loss	Throughput
0 ms	0.96 ms	2.25 ms	45.79 ms	1.22 ms	0	1614 M/s
1 ms	0.75 ms	3.63 ms	17.07 ms	1.85 ms	0	488 M/s
2 ms	0.76 ms	3.02 ms	14.29 ms	1.74 ms	0	330 M/s
3 ms	0.87 ms	3.62 ms	42.71 ms	2.49 ms	0	236 M/s
5 ms	0.76 ms	1.76 ms	8.29 ms	0.76 ms	0	150 M/s
10 ms	0.90 ms	1.81 ms	8.65 ms	0.80 ms	0	78 M/s
50 ms	0.97 ms	1.90 ms	4.66 ms	0.60 ms	0	18 M/s
100 ms	0.98 ms	2.05 ms	8.80 ms	1.30 ms	0	9 M/s

Table 7: Single publisher MQTT stress test results (QoS 2)

With a QoS level of 2, shown in Table 7, there is the benefit of guaranteed delivery to at least one *subscriber* but most importantly the throughput figures and latency are perfect for the use cases of the proposed system. (Ng et al., 2012)

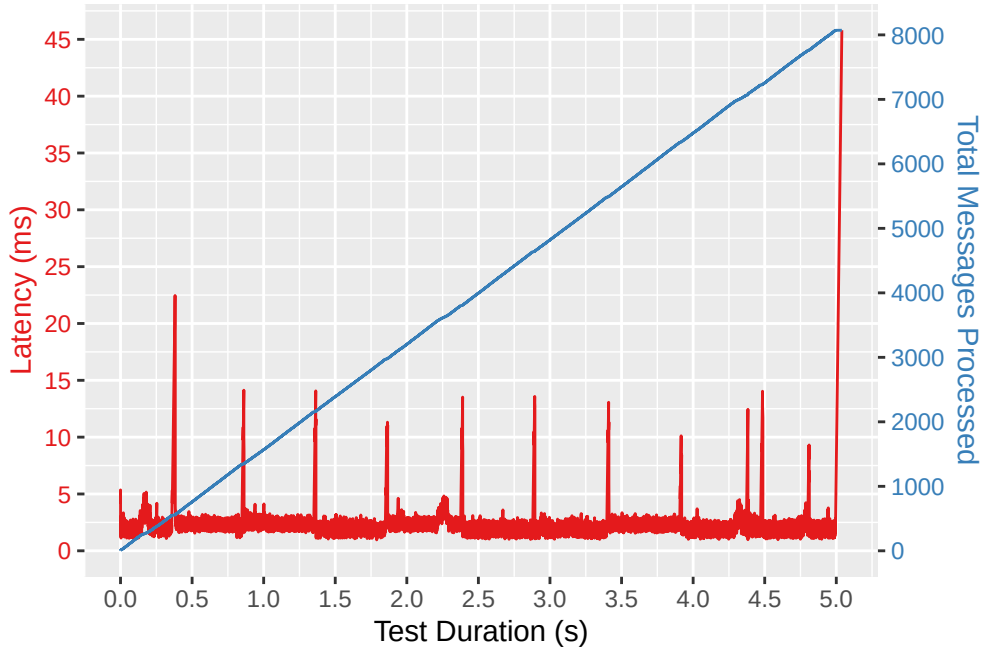

Figure 31: Latency and total processed messages over test duration for a single publisher without any delay between messages with a QoS level of 2.

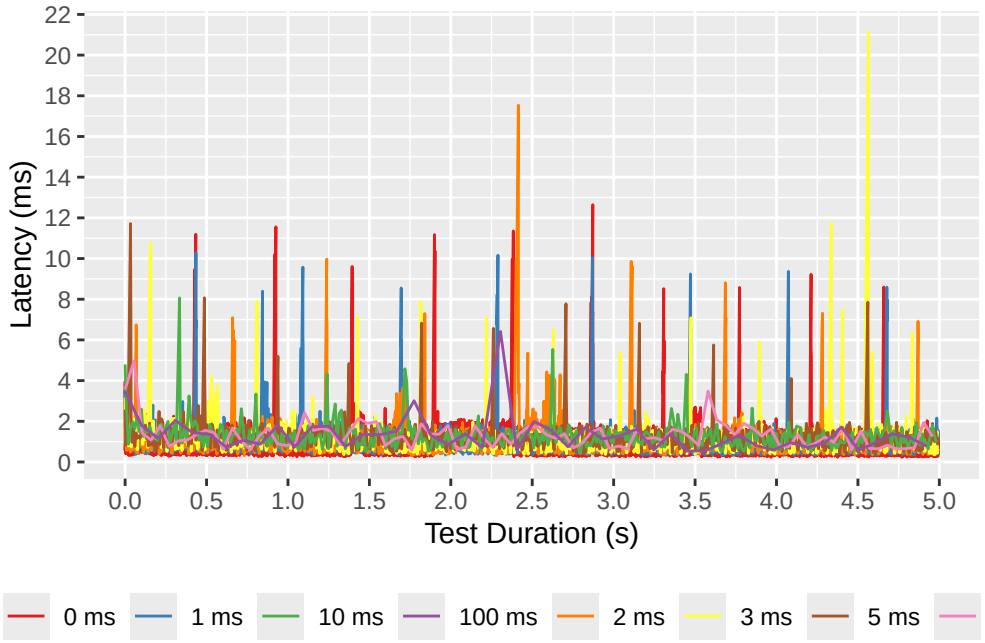

Figure 32: Latency over test duration for a single publisher with delays between messages and a QoS level of 2.

With the figures measured from these tests, it's possible to be confident that the proposed system won't be incurring any noticeable latency in any application and allows breathing room for latencies incurred from the communication medium such as wireless networks to not affect the performance of the application in a noticeable way.

Another factor to keep in mind is that on some use cases a single *subscriber* will be aggregating and acting upon messages from multiple *publishers*, something that is very common with IoT solutions and as these systems become more commonplace the number of devices involved in a system will inevitably grow as new ideas of innovative digital experiences are conceived. Given the callback-based implementation of the chosen libraries, where every single received topic will be processed by the same, manually-controlled, method, there's no overhead from subscribing to multiple topics, it's possible to extrapolate the approximate performance based on the throughput figures found on each test since for the subscriber application the effect will be the same, only from the perspective of the broker that may be a difference given the multiple TCP connections being handled.

5.4 Conclusions from the tests

With the tests that were conducted it's safe to assume that the choice of MQTT as the backbone protocol of the proposed framework was correct and its performance will allow this system to be used in any creative application where physical input or output devices are needed, even when very non-noticeable latency and guarantee of message delivery are a requirement for a pleasurable user experience.

An application developer that implements this system can be certain that even in highly distributed environments, with many input devices sending a large amount of messages per second, the only limitations in term of latency-inducing factors are going to be limited to

the logic implemented as part of the application and the communication medium between the devices and the application.

It's important to note that the tests performed in this section were all intended to measure the raw performance of the protocol and how much of an overhead represents to the overall transmission of the data. One set of tests that wasn't performed and is important to be included in future work is the overall latency of the communication between an IoT device and the application, and comparing how different devices behave and their performance characteristics when used for latency sensitive applications.

6 Conclusions and future work

The research conducted was able to identify a gap in the available solutions in the current creative applications and Internet of Things field to provide an adequate system to bridge and integrate these systems. With the problem identified and a possible proposed solution created, a thorough test was conducted to identify the viability of the idealized solution and proved that it was, in fact, viable.

Even though the proposed framework is viable, it demonstrates that there are still areas for improvement in this field of study and future work has to be conducted in order to further explore the possibilities of the synergy area between creative applications, Extended Reality, and Internet of Things.

6.1 Conclusions

With all of the results shown so far combined with the proposals and solutions discussed throughout this research work, it's safe to say that MQTT can be used as the backbone communication protocol for connecting IoT devices and creative applications, even when latency is an important factor to take into account, the protocol is performant enough to be used in such environments and provide an acceptable level of performance.

The proposed system provides assurances in terms performance, as well as in the ease of setup for commercial applications, and solves many common issues encountered whenever IoT devices are integrated with local services that are connected to a single or multiple clients.

An important point to note is that all of the solutions provided by the proposed framework are only possible due to the overarching architecture of it, approaching the problem from all angles in order to solve them, and providing solutions that go from the end client, passing through the server, and ending at individual IoT modules. Having all of these moving pieces tightly integrated into a single framework is the only way of solving all of the disclosed issues with other solutions.

6.2 Future work

Given the limited amount of time and the proposed scope of this thesis, more work can still be conducted in this field, specially in terms of user testing and validation of both, its usage and performance testing in actual games, and of the developer experience related to its usage, specially if the framework is used in novel ways with adjacent technologies in specific use cases such as pointing devices for games, position tracking, and control of real output devices in live productions, exhibits or events.

It's also important to note that due to the framework's generalized architecture aimed at solving for common IoT related issues, there's still room for research on its usage outside of the creative application space and into the fields of industrial automation and control, and home or commercial automation.

Because of the latency sensitive nature of the proposed research, there was a need to tightly couple the IoT device's firmware to the framework itself, ensuring the tightest possible loop, although future research should be conducted into an extension to the proposed work that may be built to provide a *middleware* to integrate *off-the-shelf* IoT

devices with the proposed framework.

Another area that may require more research to be conducted is in a detailed comparisons of the two major IoT communication protocols, MQTT and CoAP, in terms of raw performance for applications such as the one proposed.

References

Aiersilan, A. (2023, December 15). *Literature Review of Mixed Reality Research*. arXiv: 2312.02995 [cs]. https://doi.org/10.48550/arXiv.2312.02995

Alsop, T. (2022, February 8). *Reported price of leading consumer VR headsets 2019*. Statista. Retrieved April 28, 2024, from https://www.statista.com/statistics/1096886/reported-price-of-leading-consumer-vr-headsets-by-device/

Alsop, T. (2023, March 9). *XR market size worldwide 2021-2026*. Statista. Retrieved April 28, 2024, from https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/

Alsop, T. (2024a, February 9). *VR headset average price 2018-2028*. Statista. Retrieved June 21, 2024, from https://www.statista.com/forecasts/1338351/vr-headset-average-price-worldwide

Alsop, T. (2024b, May 14). *AR software B2C market size 2019-2029*. Statista. Retrieved June 21, 2024, from https://www.statista.com/forecasts/1337258/ar-software-b2c-market-revenue-worldwide

Andrade, A. (2015). Game engines: A survey. *EAI Endorsed Transactions on Serious Games*, *"2"*(6). Retrieved June 17, 2024, from https://eudl.eu/doi/10.4108/eai.5-11-2015.150615

Apache Software Foundation. (2024, June 3). *ActiveMQ*. Retrieved June 15, 2024, from https://activemq.apache.org/

Apple. (2023, June 5). *Introducing Apple Vision Pro: Apple's first spatial computer*. Apple Newsroom. Retrieved June 21, 2024, from https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/

Apple. (2024). *Adjust your level of immersion when using Apple Vision Pro*. Apple Support. Retrieved June 21, 2024, from https://support.apple.com/en-gb/guide/apple-vision-pro/tan899d290e4/visionos

Bahga, A., & Madisetti, V. (2014, August 9). *Internet of Things: A Hands-On Approach*. VPT.

Bandyopadhyay, S., & Bhattacharyya, A. (2013). Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. *2013 International Conference on Computing, Networking and Communications (ICNC)*, 334–340. https://doi.org/10.1109/ICCNC.2013.6504105

Baradaran Rahimi, F., Boyd, J. E., Eiserman, J. R., Levy, R. M., & Kim, B. (2022). Museum beyond physical walls: An exploration of virtual reality-enhanced experience in an exhibition-like space. *Virtual Reality*, *26*(4), 1471–1488. https://doi.org/10.1007/s10055-022-00643-5

Barker, S. K., & Shenoy, P. (2010). Empirical evaluation of latency-sensitive application performance in the cloud. *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, 35–46. https://doi.org/10.1145/1730836.1730842

BBC. (n.d.). *Micro:bit Educational Foundation*. Retrieved June 20, 2024, from https://microbit.org/

Belli, L., Cirani, S., Ferrari, G., Melegari, L., & Picone, M. (2015). A Graph-Based Cloud Architecture for Big Stream Real-Time Applications in the Internet of Things. In G. Ortiz & C. Tran (Eds.), *Advances in Service-Oriented and Cloud Computing* (pp. 91–105). Springer International Publishing. https://doi.org/10.1007/978-3-319-14886-1_10

Bevywise. (n.d.). *MQTT Broker to build enterprise IoT Applications with AI/ML*. Retrieved June 15, 2024, from https://www.bevywise.com/mqtt-broker/

Beyer, D., Löwe, S., & Wendler, P. (2019). Reliable benchmarking: Requirements and solutions. *International Journal on Software Tools for Technology Transfer*, *21*(1), 1–29. https://doi.org/10.1007/s10009-017-0469-y

Blom, J. (2015, January 5). *Exploring the Protocols of IoT - News - SparkFun Electronics*. Retrieved June 18, 2024, from https://www.sparkfun.com/news/1705

Blum, R. (2003, August 6). *Network Performance Open Source Toolkit: Using Netperf, tcptrace, NISTnet, and SSFNet*. John Wiley & Sons.

Buccafurri, F., de Angelis, V., & Lazzaro, S. (2023). MQTT-A: A Broker-Bridging P2P Architecture to Achieve Anonymity in MQTT. *IEEE Internet of Things Journal*, *10*(17), 15443–15463. https://doi.org/10.1109/JIOT.2023.3264019

Burdea, G. C., & Coiffet, P. (2003, June 30). *Virtual Reality Technology*. John Wiley & Sons.

ClassVR. (2022, September 7). *Benefits of Augmented Reality in Education*. ClassVR. Retrieved June 18, 2024, from https://www.classvr.com/blog/benefits-of-augmented-reality-in-education/

Clement, J. (2024, March 4). *Global Pokémon Go downloads 2023*. Statista. Retrieved April 28, 2024, from https://www.statista.com/statistics/641690/pokemon-go-number-of-downloads-worldwide/

Coherent Market Insights. (2024, February). *Arduino Compatible Market Size & Share Analysis - Industry Research Report - Growth Trends*. Retrieved June 20, 2024, from https://www.coherentmarketinsights.com/industry-reports/arduino-compatible-market

Connor, A. M. (2020). Creative Technologies: A Retrospective. Retrieved June 18, 2024, from https://hdl.handle.net/10292/13417

Cremona, C., & Kavakli, M. (2023). The Evolution of the Virtual Production Studio as a Game Changer in Filmmaking. In A. L. Brooks (Ed.), *Creating Digitally: Shifting Boundaries: Arts and Technologies—Contemporary Applications and Concepts* (pp. 403–429). Springer International Publishing. https://doi.org/10.1007/978-3-031-31360-8_14

Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., & Hart, J. C. (1992). The CAVE: Audio visual experience automatic virtual environment. *Communications of the ACM*, *35*(6), 64–72. https://doi.org/10.1145/129888.129892

Dallinger, L. (2023, May 25). *Mastering MQTT Packet: A usage example guide*. Cedalo. Retrieved June 18, 2024, from https://cedalo.com/blog/mqtt-packet-guide/

Doucet, L., & Pecorella, A. (2021, September 2). *Game engines on Steam: The definitive breakdown*. Retrieved April 28, 2024, from https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown

Eclipse Foundation. (2018, January 8). *Eclipse Mosquitto*. Eclipse Mosquitto. Retrieved June 15, 2024, from https://mosquitto.org/

Eclipse Foundation. (2024, February 10). *Paho-mqtt: MQTT version 5.0/3.1.1 client class* (Version 2.1.0). Retrieved June 15, 2024, from http://eclipse.org/paho

EMQ Technologies. (n.d.). *EMQX: The #1 MQTT Platform for IoT, IIoT and Connected Cars*. www.emqx.com. Retrieved June 15, 2024, from https://www.emqx.com/en

EMQX. (2020, July 20). *Introduction to MQTT 5.0 Protocol-QoS (Quality of Service)*. Medium. Retrieved June 17, 2024, from https://emqx.medium.com/introduction-to-mqtt-5-0-protocol-qos-quality-of-service-e6d9b0aaf9fb

EMQX. (2022, August 24). *MQTT over QUIC: Next-Generation IoT Standard Protocol*. www.emqx.com. Retrieved May 4, 2024, from https://www.emqx.com/en/blog/mqtt-over-quic

EMQX Team. (2023, August 25). *Using MQTT in Unity with M2MqttUnity Library: A Step-by-Step Guide*. www.emqx.com. Retrieved June 15, 2024, from https://www.emqx.com/en/blog/using-mqtt-in-unity-with-m2mqttunity-library-a-step-by-step-guide

Epic Games. (n.d.). *Unreal Engine*. Unreal Engine. Retrieved April 28, 2024, from https://www.unrealengine.com/en-US/home

Espinosa-Aranda, J. L., Vallez, N., Sanchez-Bueno, C., Aguado-Araujo, D., Bueno, G., & Deniz, O. (2015). Pulga, a tiny open-source MQTT broker for flexible and secure IoT deployments. *2015 IEEE Conference on Communications and Network Security (CNS)*, 690–694. https://doi.org/10.1109/CNS.2015.7346889

Espressif. (2023a). *FreeRTOS Overview - ESP32 - — ESP-IDF Programming Guide latest documentation*. Retrieved June 20, 2024, from https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

Espressif. (2023b, September 22). *Espressif Leads the IoT Chip Market with Over 1 Billion Shipments Worldwide | Espressif Systems*. Retrieved June 20, 2024, from https://www.espressif.com/en/news/1_Billion_Chip_Sales

Eugster, S. A. (2018, July 6). *Exchanged packets of an MQTT connection with QoS = 0*. Retrieved May 4, 2024, from https://commons.wikimedia.org/wiki/File:MQTT_protocol_example_without_QoS.svg

EveryMac. (2019). *MacBook Pro "Core i9" 2.4 16" 2019 Specs (2019 16", BTO/CTO, MacBookPro16,1, A2141, 3347)*. Retrieved June 13, 2024, from https://everymac.com/systems/apple/macbook_pro/specs/macbook-pro-core-i9-2.4-eight-core-16-2019-scissor-specs.html

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Retrieved June 18, 2024, from https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fischer, M., Rosenberg, J., Leuze, C., Hargreaves, B., & Daniel, B. (2023). The Impact of Occlusion on Depth Perception at Arm's Length. *IEEE Transactions on Visualization and Computer Graphics*, *29*(11), 4494–4502. https://doi.org/10.1109/TVCG.2023.3320239

Fleck, P., Schmalstieg, D., & Arth, C. (2020). Creating IoT-ready XR-WebApps with Unity3D. *Proceedings of the 25th International Conference on 3D Web Technology*, 1–7. https://doi.org/10.1145/3424616.3424691

Gepp, M. (2017, November 8). *HTC VIVE Announces VIVE Arts Program*. Retrieved June 17, 2024, from https://blog.vive.com/us/htc-vive-announces-vive-arts-program/

Gleasure, R., & Feller, J. (2016). A Rift in the Ground: Theorizing the Evolution of Anchor Values in Crowdfunding Communities through the Oculus Rift Case Study. *Journal of the Association for Information Systems*, *17*(10). https://doi.org/10.17705/1jais.00439

Gupta, R. (2015, February 12). *MQTT v3.1 and MQTT v3.1.1 Differences WD-01*. Retrieved May 4, 2024, from https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=55095

Haag, S., & Anderl, R. (2018). Digital twin – Proof of concept. *Manufacturing Letters*, *15*, 64–66. https://doi.org/10.1016/j.mfglet.2018.02.006

Han, Y., Niyato, D., Leung, C., Kim, D. I., Zhu, K., Feng, S., Shen, X., & Miao, C. (2023). A Dynamic Hierarchical Framework for IoT-Assisted Digital Twin Synchronization in the Metaverse. *IEEE Internet of Things Journal*, *10*(1), 268–284. https://doi.org/10.1109/JIOT.2022.3201082

Heater, B. (2024, February 27). *HTC Vive became an enterprise product while you weren't looking*. TechCrunch. Retrieved June 17, 2024, from https://techcrunch.com/2024/02/27/htc-vive-became-an-enterprise-product-while-you-werent-looking/

HiveMQ. (2024, June 14). *Hivemq/hivemq-community-edition*. Retrieved June 15, 2024, from https://github.com/hivemq/hivemq-community-edition

Holtmann, C., & Wernike, S. (2023). Extended Reality Authoring System for Creating Immersive Experiences: A Requirements Analysis. *Proceedings of the 20th International Conference on Culture and Computer Science: Code and Materiality*, 1–9. https://doi.org/10.1145/3623462.3624632

Hussain, A., Shakeel, H., Hussain, F., Uddin, N., & Ghouri, T. (2020). Unity Game Development Engine: A Technical Survey. *University of Sindh Journal of Information and Communication Technology*, *4*.

IBM & Eurotech. (2010). *MQTT V3.1 Protocol Specification*. Retrieved May 4, 2024, from https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

IBM Podcast Transcript. (2011, November). Retrieved May 4, 2024, from https://web.archive.org/web/20230306045543/https://www.ibm.com/podcasts/software/websphere/connectivity/piper_diaz_nipper_mq_tt_11182011.pdf

Jota, R., Ng, A., Dietz, P., & Wigdor, D. (2013). How fast is fast enough? a study of the effects of latency in direct-touch pointing tasks. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2291–2300. https://doi.org/10.1145/2470654.2481317

Kavanagh, S., Luxton-Reilly, A., Wuensche, B., & Plimmer, B. (2017). A systematic review of Virtual Reality in education. *Themes in Science and Technology Education*, *10*(2), 85–119. Retrieved June 18, 2024, from https://www.learntechlib.org/p/182115/

Kim, S.-M., Choi, H.-S., & Rhee, W.-S. (2015). IoT home gateway for auto-configuration and management of MQTT devices. *2015 IEEE Conference on Wireless Sensors (ICWiSe)*, 12–17. https://doi.org/10.1109/ICWISE.2015.7380346

Komianos, V. (2022). Immersive Applications in Museums: An Analysis of the Use of XR Technologies and the Provided Functionality Based on Systematic Literature Review. *JOIV : International Journal on Informatics Visualization*, *6*(1), 60–73. https://doi.org/10.30630/joiv.6.1.708

Li, W., & Huang, X. (2023). A New Way to Experience Art: Experience Design and Strategies for Immersive Exhibitions. In M. Rauterberg (Ed.), *Culture and Computing* (pp. 136–149). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-34732-0_10

Liu, M., Fang, S., Dong, H., & Xu, C. (2021). Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, *58*, 346–361. https://doi.org/10.1016/j.jmsy.2020.06.017

Long, M., & Gutwin, C. (2019). Effects of Local Latency on Game Pointing Devices and Game Pointing Tasks. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/10.1145/3290605.3300438

Longo, E., Redondi, A. E., Cesana, M., Arcia-Moret, A., & Manzoni, P. (2020). MQTT-ST: A Spanning Tree Protocol for Distributed MQTT Brokers. *ICC 2020 - 2020*

IEEE International Conference on Communications (ICC), 1–6. https://doi.org/10.1109/ICC40277.2020.9149046

M5Stack. (n.d.). *Modular Rapid ESP32 IoT Development Board - ESP32 dev kits*. M5Stack. Retrieved June 20, 2024, from https://m5stack.com/

Machkovech, S. (2016, January 28). *HTC Vive Pre impressions: A great VR system has only gotten better*. Ars Technica. Retrieved June 21, 2024, from https://arstechnica.com/gaming/2016/01/htc-vive-pre-impressions-a-great-vr-system-has-only-gotten-better/

Mathews, C. C., & Wearn, N. (2016). How Are Modern Video Games Marketed? *The Computer Games Journal*, *5*(1), 23–37. https://doi.org/10.1007/s40869-016-0023-2

McCann, J., & Pollard, N. (2007). Responsive characters from motion fragments. *ACM SIGGRAPH 2007 Papers*, 6–es. https://doi.org/10.1145/1275808.1276385

Memomi. (n.d.). *Augmented Reality & Artificial Intelligence Mirror Software*. Memomi. Retrieved June 18, 2024, from https://memorymirror.com/

Mishra, B., & Kertesz, A. (2020). The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access*, *8*, 201071–201086. https://doi.org/10.1109/ACCESS.2020.3035849

Mishra, B., Mishra, B., & Kertesz, A. (2021). Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements. *Energies*, *14*(18), 5817. https://doi.org/10.3390/en14185817

Moi, T., Cibicik, A., & Rølvåg, T. (2020). Digital twin based condition monitoring of a knuckle boom crane: An experimental study. *Engineering Failure Analysis*, *112*, 104517. https://doi.org/10.1016/j.engfailanal.2020.104517

MQTT.org. (2015, March 15). *10th Birthday Party | MQTT*. Retrieved May 4, 2024, from https://web.archive.org/web/20150315025826/https://mqtt.org/2009/07/10th-birthday-party

MQTT.org. (2018, April 13). *Differences between 3.1.1 and 5.0*. GitHub. Retrieved May 4, 2024, from https://github.com/mqtt/mqtt.org/wiki/Differences-between-3.1.1-and-5.0

MQTT.org. (2022a). *MQTT - The Standard for IoT Messaging*. Retrieved May 4, 2024, from https://mqtt.org/

MQTT.org. (2022b). *MQTT: Use Cases*. Retrieved May 4, 2024, from https://mqtt.org/use-cases/

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Systems Engineering Symposium (ISSE)*, 1–7. https://doi.org/10.1109/SysEng.2017.8088251

Ng, A., Lepinski, J., Wigdor, D., Sanders, S., & Dietz, P. (2012). Designing for low-latency direct-touch input. *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, 453–464. https://doi.org/10.1145/2380116.2380174

Nishikawa-Pacher, A. (2022). Research Questions with PICO: A Universal Mnemonic. *Publications*, *10*(3), 21. https://doi.org/10.3390/publications10030021

Notes, V. (2021, September 1). *What are HTTP requests and Spring MVC?* Javarevisited. Retrieved June 18, 2024, from https://medium.com/javarevisited/here-we-will-discuss-dab31e2440b2

Nuiaa, R. R., Manickam, S., & Alsaeedi, A. (2021). Distributed reflection denial of service attack: A critical review. *11*. https://doi.org/10.11591/ijece.v11i6.pp%25p

OASIS. (2019, March 7). *MQTT Version 5.0*. Retrieved May 4, 2024, from http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Octavo Labs. (n.d.). *VerneMQ - A MQTT broker that is scalable, enterprise ready, and open source*. Retrieved June 15, 2024, from https://vernemq.com/

Oculus. (2024, May 31). *Oculus Integration (Deprecated) | Integration | Unity Asset Store*. Retrieved June 20, 2024, from https://assetstore.unity.com/packages/tools/integration/oculus-integration-deprecated-82022

Pallets. (2024, June 6). *Welcome to Flask — Flask Documentation (3.0.x)*. Retrieved June 20, 2024, from https://flask.palletsprojects.com/en/3.0.x/

Perera, S., Barnes, D., & Zelinsky, D. (2014). Exploration: Simultaneous Localization and Mapping (SLAM). In K. Ikeuchi (Ed.), *Computer Vision: A Reference Guide* (pp. 268–275). Springer US. https://doi.org/10.1007/978-0-387-31439-6_280

Perkins, C., & Jagannadh, T. (1995). DHCP for mobile networking with TCP/IP. *Proceedings IEEE Symposium on Computers and Communications*, 255–261. https://doi.org/10.1109/SCAC.1995.523675

Peters, J. (2021, November 18). *Kid A Mnesia Exhibition is an unsettling and beautiful Radiohead art exhibit*. The Verge. Retrieved June 18, 2024, from https://www.theverge.com/22788135/radiohead-kid-a-mnesia-exhibition-review

Phanekham, D., & Jones, R. (2020, June 17). *Using netperf and ping to measure network latency*. Google Cloud Blog. Retrieved June 13, 2024, from https://cloud.google.com/blog/products/networking/using-netperf-and-ping-to-measure-network-latency

Pierce, D. (2023, September 27). *Meta's $499.99 Quest 3 headset is all about mixed reality and video games*. The Verge. Retrieved June 21, 2024, from https://www.theverge.com/2023/9/27/23889059/meta-quest-3-headset-mixed-reality-price-release-date

Porter, J. (2023, November 29). *HTC's Vive Ultimate Trackers are a sleeker way to keep tabs on your VR limbs and other objects*. The Verge. Retrieved June 17, 2024, from https://www.theverge.com/2023/11/29/23980817/htc-vive-ultimate-tracker-object-limb-tracking-vr-xr-elite

Python. (2024, June 14). *Welcome to Python.org*. Python.org. Retrieved June 20, 2024, from https://www.python.org/

Radiohead. (2021, November 18). *Radiohead - KID A MNESIA EXHIBITION*. Retrieved June 18, 2024, from https://kida-mnesia.com

Raj, T., Hashim, F. H., Huddin, A. B., Ibrahim, M. F., & Hussain, A. (2020). A Survey on LiDAR Scanning Mechanisms. *Electronics*, *9*(5), 741. https://doi.org/10.3390/electronics9050741

Rauschnabel, P. A., Felix, R., Hinsch, C., Shahab, H., & Alt, F. (2022). What is XR? Towards a Framework for Augmented and Virtual Reality. *Computers in Human Behavior*, *133*, 107289. https://doi.org/10.1016/j.chb.2022.107289

Rosenberg, L. B. (2022). Augmented Reality: Reflections at Thirty Years. In K. Arai (Ed.), *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 1* (pp. 1–11). Springer International Publishing. https://doi.org/10.1007/978-3-030-89906-6_1

Rubin, P. (2014, May 20). *The Inside Story of Oculus Rift and How Virtual Reality Became Reality | WIRED*. Retrieved June 21, 2024, from https://www.wired.com/2014/05/oculus-rift-4/

Salim, F., & Haque, U. (2015). Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical

systems, and Internet of Things. *International Journal of Human-Computer Studies*, *81*, 31–48. https://doi.org/10.1016/j.ijhcs.2015.03.003

Shelby, Z., Hartke, K., & Bormann, C. (2014, June). *The Constrained Application Protocol (CoAP)* (Request for Comments No. RFC 7252). Internet Engineering Task Force. https://doi.org/10.17487/RFC7252

Silva, M., & Teixeira, L. (2022). eXtended Reality (XR) Experiences in Museums for Cultural Heritage: A Systematic Review. In Z. Lv & H. Song (Eds.), *Intelligent Technologies for Interactive Entertainment* (pp. 58–79). Springer International Publishing. https://doi.org/10.1007/978-3-030-99188-3_5

Smithsonian. (n.d.). *Bone Hall | Smithsonian National Museum of Natural History*. Retrieved June 18, 2024, from https://naturalhistory.si.edu/exhibits/bone-hall

Sodazot, I. (**typedirector**). (2015, March 1). *Quantum Space / interactive room* [Video]. Retrieved June 18, 2024, from https://vimeo.com/120944206

Sourin, A. (2017). Case Study: Shared Virtual and Augmented Environments for Creative Applications. In R. Earnshaw (Ed.), *Research and Development in the Academy, Creative Industries and Applications* (pp. 49–64). Springer International Publishing. https://doi.org/10.1007/978-3-319-54081-8_5

Sun, X., Ansari, N., & Wang, R. (2016). Optimizing Resource Utilization of a Data Center. *IEEE Communications Surveys & Tutorials*, *18*(4), 2822–2846. https://doi.org/10.1109/COMST.2016.2558203

Sutherland, I. E. (1968). A head-mounted three dimensional display. *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, 757–764. https://doi.org/10.1145/1476589.1476686

Svanæs, D., Scharvet Lyngby, A., Bärnhold, M., Røsand, T., & Subramanian, S. (2021). UNITY-Things: An Internet-of-Things Software Framework Integrating Arduino-Enabled Remote Devices with the UNITY Game Engine. In X. Fang (Ed.), *HCI in Games: Experience Design and Game Mechanics* (pp. 378–388). Springer International Publishing. https://doi.org/10.1007/978-3-030-77277-2_29

Thomsen, M. (2010, February 23). *History of the Unreal Engine*. IGN. Retrieved June 17, 2024, from https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

TIOBE. (2024, June). *TIOBE Index*. TIOBE. Retrieved June 20, 2024, from https://www.tiobe.com/tiobe-index/

Unity Technologies. (n.d.). *Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine*. Unity. Retrieved April 28, 2024, from https://unity.com/

Unity Technologies. (2024, June 19). *Unity - Manual: UnityEvents*. Retrieved June 20, 2024, from https://docs.unity3d.com/Manual/UnityEvents.html

Vaccari, I., Aiello, M., & Cambiaso, E. (2020). SlowITe, a Novel Denial of Service Attack Affecting MQTT. *Sensors*, *20*(10), 2932. https://doi.org/10.3390/s20102932

VanDerHorn, E., & Mahadevan, S. (2021). Digital Twin: Generalization, characterization and implementation. *Decision Support Systems*, *145*, 113524. https://doi.org/10.1016/j.dss.2021.113524

Viganò, G. P. (2024, May 27). *Gpvigano/M2MqttUnity*. Retrieved June 15, 2024, from https://github.com/gpvigano/M2MqttUnity

W3C. (2000, May 8). *SOAP Specifications*. Retrieved June 18, 2024, from https://www.w3.org/TR/soap/

Wan, M. (2022, November 9). *Everything You Need to Know About DMX512 Control - LEDYi Lighting*. https://www.ledyilighting.com/. Retrieved June 20, 2024, from

https://www.ledyilighting.com/everything-you-need-to-know-about-dmx512-control/

Wang, Z., Han, K., & Tiwari, P. (2021). Digital Twin Simulation of Connected and Automated Vehicles with the Unity Game Engine. *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 1–4. https://doi.org/10.1109/DTPI52967.2021.9540074

Webster, A. (2017, July 6). *Pokémon Go's wild first year: A timeline*. The Verge. Retrieved June 21, 2024, from https://www.theverge.com/2017/7/6/15888210/pokemon-go-one-year-anniversary-timeline

Wedel, M., Bigné, E., & Zhang, J. (2020). Virtual and augmented reality: Advancing research in consumer marketing. *International Journal of Research in Marketing*, *37*(3), 443–465. https://doi.org/10.1016/j.ijresmar.2020.04.004

Xue, L., Parker, C. J., & Hart, C. A. (2022). How augmented reality can enhance fashion retail: A UX design perspective. *International Journal of Retail & Distribution Management*, *51*(1), 59–80. https://doi.org/10.1108/IJRDM-09-2021-0435

You, D., Seo, B.-S., Jeong, E., & Kim, D. H. (2018). Internet of Things (IoT) for Seamless Virtual Reality Space: Challenges and Perspectives. *IEEE Access*, *6*, 40439–40449. https://doi.org/10.1109/ACCESS.2018.2829194

Zero Density. (n.d.). *Home | Zero Density*. Zero Density. Retrieved June 21, 2024, from https://www.zerodensity.io/

Zhou, Z. (2023, January 12). *MQTT QoS 0, 1, 2 Explained: A Quickstart Guide*. www.emqx.com. Retrieved June 17, 2024, from https://www.emqx.com/en/blog/introduction-to-mqtt-qos