A picture containing logo

Description automatically generated**Master of Science in Embedded Systems Design**

**Project’s name:**

Mobile platform power management system with Fuel-cell

**Under the guidance of**

**Prof. Oliver Prenzel**

**Submitted by**

**Hossein Baghaei (39383)**

**Submitted on**

**04.01.2023**

Contents

[1 Introduction 3](#_Toc123733187)

[2 Programming languages 3](#_Toc123733188)

[2.1 Embedded-C 3](#_Toc123733189)

[2.2 Python 3](#_Toc123733190)

[3 Compilers 4](#_Toc123733191)

[3.1 Arduino 4](#_Toc123733192)

[3.2 PyCharm Edu 4](#_Toc123733193)

[4 Libraries 5](#_Toc123733194)

[4.1 **Libraries which are used for programming by Embedded-C:** 5](#_Toc123733195)

[4.2 **Libraries which are used in Python** 5](#_Toc123733196)

[5 MCU 6](#_Toc123733197)

[6 Connection between ESP32 and GUI 6](#_Toc123733198)

[7 Which data are sending, and which data are receiving 6](#_Toc123733199)

[8 MQTT( MQ Telemetry Transport) 7](#_Toc123733200)

[**8.1** **How does MQTT work** 7](#_Toc123733201)

[**8.2** **what is MQTT broker** 8](#_Toc123733202)

[**8.3** **MQTT message types** 8](#_Toc123733203)

[**8.4** **Advantages of using MQTT** 10](#_Toc123733204)

[**8.5** **Flaws of MQTT** 11](#_Toc123733205)

[**8.6** **Applications and use cases of the MQTT protocol** 11](#_Toc123733206)

[**8.7** **MQTT and Internet of Things** 13](#_Toc123733207)

[**8.8** **levels of service quality** 14](#_Toc123733208)

[**8.9** **Historical versions of the MQTT protocol** 15](#_Toc123733209)

[**8.10** **changes to MQTT** 16](#_Toc123733210)

[9 References 17](#_Toc123733211)

# **Introduction**

This report informs reader about the communication between ESP32 and GUI. For this portion of project, the MQTT protocol is used for transferring data between ESP32 and GUI. The programming languages which are used in this project are Embedded-C and Python which they are explained clearly below.

# **Programming languages**

## Embedded-C

The C Standards Committee created Embedded C as a set of language extensions for the C programming language to solve issues of commonality between C extensions for various embedded devices.

In order to implement more advanced microprocessor features like fixed-point arithmetic, many different memory banks, and fundamental I/O operations, embedded C programming often necessitates nonstandard additions to the C language. A Technical Report created by the C Standards Committee, which was most recently revised in 2008 and revisited in 2013, established a uniform standard that all implementations must follow. It offers a variety of features not found in standard C, such as named address spaces, fixed-point arithmetic, and fundamental I/O hardware addressing. The majority of conventional C's syntax and semantics are used by embedded C, including the main() function, variable definitions, datatype declarations, conditional statements (if, switch case), loops (while, for), functions, arrays and strings, structures and unions, bit operations, macros, and more.

## Python

A high-level, all-purpose programming language is Python. Code readability is prioritized in its design philosophy, which makes heavy use of indentation.

Python uses garbage collection and has dynamic typing. It supports a variety of programming paradigms, including procedural, object-oriented, and functional programming as well as structured programming (especially this). Due to its extensive standard library, it is frequently referred to as a "batteries included" language.

# **Compilers**

## Arduino

Arduino is a firm that creates and produces single-board microcontrollers and microcontroller kits for creating digital devices. It is an open-source hardware and software initiative. Its hardware products are available under a CC BY-SA license, while software is licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), permitting the fabrication of Arduino boards and software distribution by anybody. Commercial Arduino boards are offered on the official website or from accredited distributors.

Different types of microprocessors and controllers are used in Arduino board designs. The boards have a variety of extension boards (called "shields"), breadboards (for prototyping), and other circuits that can be interfaced to the sets of digital and analog input/output (I/O) pins on the boards. The boards have serial communications interfaces, some of which support USB (Universal Serial Bus), which are also used to load programs. The C and C++ programming languages, as well as a standard API known as the Arduino Programming Language, which was modeled after the Processing language and used with a modified version of the Processing IDE, can be used to program the microcontrollers. The Arduino project offers an integrated development environment (IDE) and a command line tool created in Go in addition to using conventional compiler toolchains.

## PyCharm Edu

Python programming is done using the integrated development environment (IDE) PyCharm. In addition to supporting Django web development, it offers code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and more. The JetBrains firm in the Czech Republic creates PyCharm.

It is cross-platform and functional with Linux, macOS, and Microsoft Windows. There is a Community Edition of PyCharm that is distributed under the Apache License and a Professional Edition that is distributed under a proprietary license. Less features are included in PyCharm Community Edition than in Professional Edition.

# **Libraries**

## **Libraries which are used for programming by Embedded-C:**

For communication between ESP32 and GUI, I used two libraries named WiFi.h which are used for connecting ESP32 to a Wi-Fi modem and PubSubClient.h for publishing and subscribing data between ESP32 and GUI with MQTT protocol.

## **Libraries which are used in Python**

Libraries and functions which are used mentioned below:

Matplotlib, time, paho.mqtt, sys, datetime, os, json, tkinter and numpy which are explained respectively.

**Matplotlib**: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

**Time**: time() Function. In Python, the time() function returns the number of seconds passed since epoch (the point where time begins). For the Unix system, January 1, 1970, 00:00:00 at UTC is epoch. This function is used for making delay.

**paho.mqtt**: This code offers a client class that allows programs to connect to a MQTT broker in order to post messages, subscribe to topics, and receive messages that have already been published. Additionally, it offers certain assistance functions to make it simple to publish one-time messages to a MQTT server.

**Sys**:The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.

**Datetime**: datetime in Python is the combination between dates and times. The attributes of this class are similar to both date and separate classes. These attributes include day, month, year, minute, second, microsecond, hour, and tzinfo.

**Os**: The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

**Json**: JavaScript Object Notation (JSON) is a standardized format commonly used to transfer data as text that can be sent over a network. It's used by lots of APIs and Databases, and it's easy for both humans and machines to read. JSON represents objects as name/value pairs, just like a Python dictionary.

**Tkinter**: Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and macOS installs of Python. The name Tkinter comes from Tk interface.

**Numpy**: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

# **MCU**

**ESP32**: The ESP32 family of system on a chip microcontrollers features integrated Wi-Fi and dual-mode Bluetooth and is inexpensive and low power. The Tensilica Xtensa LX6 dual-core or single-core microprocessor, Tensilica Xtensa LX7 dual-core, or a single-core RISC-V microprocessor are used in the ESP32 series, which also has integrated antenna switches, RF baluns, power amplifiers, low-noise receive amplifiers, filters, and power-management modules. Chinese business Espressif Systems, with headquarters in Shanghai, invented and constructed the ESP32, which is produced by TSMC using their 40 nm technology. It is the ESP8266 microcontroller's replacement.

# **Connection between ESP32 and GUI**

They transfer data with MQTT protocol which is mentioned below and explained completely at the end of report.

MQTT is a kind of lightweight IoT messaging protocol based on the publish/subscribe model, which can provide real-time and reliable messaging service for IoT devices, only using very little code and bandwidth. It is suitable for devices with limited hardware resources and the network environment with limited bandwidth. Therefore, [MQTT protocol](https://mqtt.org/) is widely used in IoT, mobile internet, IoV, electricity power, and other industries.

# **Which data are sending, and which data are receiving**

For this part, Battery’s voltage and current, Fuel-cell’s voltage and current and SOC are transferring to GUI and four commands return from GUI to ESP32 for turning off fuel-cell’s PMC, turning on fuel-cell’s PMC, turning off battery’s PMC and turning on battery’s PMC manually.

# **MQTT( MQ Telemetry Transport)**

A lightweight open messaging protocol called MQTT (MQ Telemetry Transport) gives network clients with limited resources an easy way to share telemetry data in low-bandwidth settings. The protocol is used for machine-to-machine (M2M) communication and uses a publish/subscribe communication structure.

MQTT was developed as a low-overhead protocol to work around bandwidth and CPU restrictions. It was intended to operate in an embedded setting where it could offer a dependable, efficient way for communication. MQTT is a suitable option for wireless networks that face different levels of latency owing to sporadic bandwidth limits or unreliable connections because it connects devices with a small code footprint. The protocol is used in a variety of sectors, including telecommunications, energy, and the automotive industry.

While the MQ in MQTT refers to a product named IBM MQ, the TT in MQTT stands for Telemetry Transport. Although Message Queuing Telemetry Transport is occasionally used as the spell-out for MQTT, message queuing is not used in MQTT communication.

## **How does MQTT work**

MQTT's publish/subscribe (pub/sub) communication model is an alternative to conventional client-server architecture that communicates directly with an endpoint with the goal of optimizing the available bandwidth. The client that transmits a message (the publisher), in contrast, is separated from the client or clients that receive the messages in the pub/sub paradigm (or the subscribers). Since neither the publishers nor the subscribers communicate with one another directly, the brokers handle their connections on their behalf.

Publishers and subscribers are terminology used to describe different types of MQTT clients, depending on whether they are actively posting messages or have subscribed to receive them. The same MQTT client can perform both of these tasks. A publish occurs when a device (or client) wishes to communicate data to a server (or broker). A subscription is what is used when the process is performed backwards. Multiple clients can connect to a broker and subscribe to subjects they are interested in under the pub/sub model.

When a broker's connection with a client that is subscribing breaks, the broker will buffer messages and send them to the client once the connection is restored. The broker may close the connection and send subscribers a cached message with instructions from the publisher if the connection between the publishing client and the broker is abruptly severed.

The pub/sub approach is described in this IBM writeup: "Publishers send the messages, interested subscribers receive them, and brokers forward the publishers' messages to the subscribers. MQTT clients, such as publishers and subscribers, can only speak to a MQTT broker. MQTT clients can be any hardware or software that executes a MQTT library, ranging from microcontrollers like the Arduino to complete application servers housed in the cloud."

## **what is MQTT broker**

The clients that send messages and the subscribers who receive them are connected through a MQTT broker. The broker is the post office itself in the analogy of the post office. Before being delivered to the subscriber, all messages must pass via the broker.

When choosing a MQTT broker, businesses should evaluate them based on their scalability, integration, monitoring, and failure-resistance characteristics. Brokers may have to manage millions of concurrently connected MQTT clients.

## **MQTT message types**

Connection, authentication, communication, and termination are the four phases of a MQTT session. In order to connect to the broker, a client first establishes a Transmission Control Protocol/Internet Protocol (TCP/IP) connection using either a normal port or a special port that has been set by the broker's operators. It is crucial to be aware when setting up the connection because if the server receives a client identity that has already been used, an existing session may be continued.

For non-encrypted communication, the usual ports are 1883 and 8883, respectively, but for encrypted communication, Secure Sockets Layer (SSL)/Transport Layer Security is used (TLS). The client verifies the server certificate and authenticates the server during the SSL/TLS handshake. During the handshake, the client can also give the broker a client certificate. This can be used by the broker to verify the client's identity. Brokers now typically enable client authentication via SSL/TLS client-side certificates, even though it is not a requirement of the MQTT specification.

SSL/TLS may not always be an option and, in some circumstances, may not be desired because the MQTT protocol aspires to be a protocol for resource-constrained and IoT devices. When this happens, authentication takes the form of a cleartext username and password that the client sends to the server as part of the CONNECT/CONNACK packet sequence. Additionally, some brokers welcome anonymous clients, particularly open brokers who are exposed online. In some circumstances, the username and password are omitted entirely.

MQTT is regarded as a lightweight protocol because each of its messages only requires a little amount of code. Each message has a fixed header that is 2 bytes in size, an optional variable header, a payload that can only include 256 MB of data, and a quality of service (QoS) level.

A client can conduct publish, subscribe, unsubscribe, and ping activities throughout the communication phase. The publish action transmits a binary data block, or "content," to a subject that the publisher specifies.

Up to 256 MB in size, MQTT supports message binary big objects (BLOBs). The content's format will depend on the application. A SUBSCRIBE/SUBACK packet pair is used to subscribe to topics, while a UNSUBSCRIBE/UNSUBACK packet pair is used to unsubscribe from topics.

Using a unique delimiter character, the forward slash (/), topic strings naturally create a topic tree. Using unique wildcard characters, a client can subscribe to and unsubscribe from entire branches of the subject tree. There are two wildcard symbols: the plus sign (+), which has a single level, and the hash symbol (#), which has multiple levels. The dollar symbol ($), a unique topic character, excludes a topic from all root wildcard subscriptions. $ is typically used to send server- or system-specific messages.

A client can also ping the broker server using a PINGREQ/PINGRESP packet sequence during the communication phase. This packet's translation is basically ARE YOU ALIVE/YES I AM ALIVE. The only purpose of this action is to keep the TCP connection active and make sure it hasn't been terminated by a gateway or router.

A publisher or subscriber can end a MQTT session by sending the broker a DISCONNECT message and then cutting the connection. Because it enables the client to quickly reconnect by entering its client identity and picking up where it left off, this is known as a graceful shutdown.

The broker may transmit subscribers a message from the publisher that the broker has previously cached if the disconnect occurs unexpectedly and the publisher does not have time to issue a DISCONNECT message. Subscribers receive instructions on what to do in the event of the publisher's untimely death in the message, which is referred to as a final will and testament.

Table

Description automatically generated

## **Advantages of using MQTT**

The MQTT protocol architecture's lightweight characteristics and little overhead help to ensure efficient data transfer with limited bandwidth and lessen the strain on the CPU and RAM. The following are a few of MQTT's advantages over rival protocols:

-low network use due to decreased data packets;

-efficient data distribution; -effective implementation of remote sensing and control;

-efficient data transfer and speedy implementation;

-delivers messages quickly and effectively;

-consumes little power, which is helpful for the devices that are linked; and

-improves network bandwidth.

## **Flaws of MQTT**

The following are possible MQTT drawbacks:

-When compared to Constrained Application Protocol, MQTT has slower transmit cycles (CoAP).

-CoAP utilizes a reliable resource discovery approach, whereas MQTT's resource discovery uses flexible topic subscription.

-MQTT lacks encryption. In its place, security encryption is performed using TLS/SSL (Transport Layer Security/Secure Sockets Layer).

-Building a MQTT network that is globally scalable is challenging.

-Authentication, interoperability, and security are further MQTT problems.

The MQTT protocol has historically been used in secure back-end networks for application-specific objectives because it was not created with security in mind. Because of how easy a large tree may be formed by MQTT's topic structure, it is unclear how to break up a tree into smaller logical domains that can be federated. Since of this, building a globally scalable MQTT network is challenging because complexity rises as the size of the topic tree does.

MQTT's lack of compatibility is another drawback. Problems can occur because message payloads are binary and lack information about their encoding, especially in open architectures where various programs from various manufacturers are meant to function harmoniously with one another.

As mentioned before, the MQTT protocol provides only a few basic authentication features. In order to use MQTT securely, usernames and passwords must be exchanged in cleartext using the SSL/TLS protocol, which is regrettably not a lightweight one.

There is no easy way in MQTT to regulate who owns a subject and who may publish content over it; instead, you must use exclusive, out-of-band methods for client-side certificate authentication. This makes it simple to intentionally or mistakenly introduce destructive messages into the network.

## **Applications and use cases of the MQTT protocol**

MQTT works well for applications needing remote monitoring because of its lightweight attributes, such as the following:

-synchronization of sensors, such as fire detectors or motion sensors for theft detection, to determine if a hazard is valid;

-monitoring health parameters using sensors for patients leaving a hospital; and

-sensors alerting people of danger.

A text-based messaging program for real-time communication is another application that makes use of MQTT's minimal data and power requirements. For instance, Facebook utilizes MQTT for its Messenger program because, in addition to preserving battery life during mobile phone-to-phone chatting, the protocol also enables messages to be efficiently sent in milliseconds despite patchy internet connections worldwide.

Diagram

Description automatically generated

MQTT is supported by the majority of big cloud service providers, such as Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure.

Applications utilizing M2M and IoT devices for real-time analytics, preventative maintenance, and monitoring in settings such as smart homes, healthcare, logistics, industry, and manufacturing are well suited for MQTT.

## **MQTT and Internet of Things**

According to MQTT.org, MQTT clients can be used on small microcontrollers because they utilize less resources and are small. MQTT headers are compact to maximize network bandwidth. Additionally, the group claims that MQTT "can scale to communicate with millions of IoT devices."

MQTT is one of the most widely used protocols in IoT and IIoT infrastructure as a result. For instance, the utilities sector uses MQTT to effectively transport data between their services and customers' and devices' devices.

Examples of MQTT use in IoT or IIoT instructure include the following:

-Smart metering. The MQTT protocol can be used to transmit data with guaranteed message delivery to provide accurate meter readings in real time. This helps make billing more accurate.

-Gathering ambient sensor data. Sensors used in remote environments are often low-power devices, so MQTT is a good fit for IoT sensor buildouts with lower-priority data transmission needs.

-Machine health data. Ably, which provides a pub/sub messaging platform, gives the example of a wind turbine requiring "guaranteed delivery of machine health data to local teams even before that information hits a data center."

-Billing systems. MQTT helps eliminate duplicate or lost message packets in billing or invoicing.

MQTT enables resource-constrained IoT devices to convey, or publish, information about a particular topic to a server that functions as a MQTT message broker, whether it be used for smart meters, quick outage response, or other IoT and IIoT applications. The broker then distributes the information to those clients who have previously subscribed to the subject.

A subject appears to a person as a hierarchical file path. Clients have the option of subscribing to a single level of a topic's hierarchy or to several levels by using a wildcard character. The client could be an application in a data center processing IoT data or an IoT sensor in the field.

The MQTT protocol is supported by various IoT systems, including Carriots, Evrythng, and ThingWorx.

## **levels of service quality**

An agreement between a message's sender and recipient is referred to as QoS. It serves as a crucial component of MQTT, allowing the client to select between three service levels.

How the content is controlled by the MQTT protocol is determined by the three different QoS levels. Although greater levels of QoS are more dependable, they also need more latency and bandwidth, therefore paying clients can choose the greatest level of QoS they want to get.

Diagram

Description automatically generated with low confidence

The unacknowledged service level is the most basic QoS level. The publisher transmits a message to the broker once, and the broker passes the message to subscribers once, in a PUBLISH packet sequence. The broker does not save the message, and there is no mechanism in place to ensure that it was received appropriately. At most once, QoS0, or fire and forget are other names for this QoS level.

Known service is the second level of quality of service. Between the publisher and its broker, as well as between the broker and subscribers, this QoS level employs a PUBLISH/PUBACK packet sequence. If an acknowledgement is not received in a timely manner, a retry mechanism will send the original information again. An acknowledgment packet certifies that content has been received. The subscriber can end up receiving many copies of the same communication as a result. This QoS level is also known as QoS1 or at least once.

A guarantee of service is the third QoS level. Two pairs of packets are used to deliver the message at this QoS level. PUBREL/PUBCOMP and PUBLISH/PUBREC are the names of the first and second pairs, respectively. The two pairings make sure that the message will only be sent once, regardless of how many times it is attempted. This QoS level is also known as QoS2 or exactly once.

## **Historical versions of the MQTT protocol**

Dr. Andy Stanford-Clark of IBM and Arlen Nipper of Arcom, now known as Eurotech, developed MQTT in 1999. MQTT was developed as a dependable and affordable solution to link remote enterprise servers to monitoring equipment used in the oil and gas sectors. They were tasked with figuring out how to push data from pipeline sensors in the desert to remote SCADA systems, and in order to reduce the cost of satellite link transmission, they chose a TCP/IP-based pub/sub topology that would be event-driven.

Although MQTT and IBM are still intimately linked, the Organization for the Advancement of Structured Information Standards now controls it as an open standard (OASIS).

MQTT is not a component of the original IBM MQSeries, despite what its name may imply; nonetheless, as of version 7.1, it is included in WebSphere MQ. The SCADA protocol, MQ Integrator SCADA Device Protocol (MQIsdp), and WebSphere MQTT (WMQTT) were all prior names for MQTT; however, all of these names are no longer in use.

Depending on the version, MQTT offers a variety of specifications. MQTT version 3.1.1 has been replaced by version 5.0. According to OASIS, some more recent requirements consist of the following:

-the use of pub/sub message patterns;

-a mechanism that can notify users when abnormal disconnections occur;

-the three levels of message delivery: at most once, at least once and exactly once;

-the minimization of transport overhead and protocol exchanges to reduce network traffic; and

-an agnostic messaging transport referring to the content of the payload.

## **changes to MQTT**

On October 28, 2015, MQTT received official approval as an OASIS standard. It was approved as an International Organization for Standardization (ISO) standard at the end of January 2016. As part of its ongoing development, the protocol now supports WebSocket, another protocol that provides real-time two-way communication between clients and brokers. The v3.1.1 standard and the v5.0 standard, both of which were accepted as OASIS standards, were two notable later revisions. Version 5.0, as one example of its changes, added enhanced error reporting, shared subscriptions, message and session expiries, metadata in message headers, and subject aliasing.

# **References**

<https://www.emqx.com/en/blog/how-to-use-mqtt-in-python>

<https://en.wikipedia.org/wiki/Python_(programming_language)>

<https://en.wikipedia.org/wiki/Embedded_C>

<https://en.wikipedia.org/wiki/Arduino>

<https://en.wikipedia.org/wiki/PyCharm>

<https://en.wikipedia.org/wiki/Matplotlib>

<https://pypi.org/project/pahomqtt/#:~:text=This%20code%20provides%20a%20client,It%20supports%20Python%202.7>.

<https://www.geeksforgeeks.org/python-sys-module/>

<https://www.simplilearn.com/tutorials/python-tutorial/datetime-in-python#:~:text=datetime%20in%20Python%20is%20the,microsecond%2C%20hour%2C%20and%20tzinfo>.

<https://www.tutorialsteacher.com/python/os-module#:~:text=The%20OS%20module%20in%20Python,with%20the%20underlying%20operating%20system>.

<https://realpython.com/lessons/what-is-json/#:~:text=JavaScript%20Object%20Notation%20(JSON)%20is,just%20like%20a%20Python%20dictionary>.

<https://en.wikipedia.org/wiki/Tkinter>

<https://en.wikipedia.org/wiki/NumPy>

<https://en.wikipedia.org/wiki/ESP32>

https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport#:~:text=MQTT%20(MQ%20Telemetry%20Transport)%20is,information%20in%20low%2Dbandwidth%20environments.