Internet of Things Communication Systems based on MQTT and BLE Protocols

Ioan NARITA, Florin POPESCU and Cristina DESPINA-STOIAN

***Abstract*—We are facing a new era of technology, where we want to connect according to the Internet of Things (IoT) concept, machine to machine, machine to infrastructure, or even machine to environment. The current paper is concerned with two of the most popular and important communication IoT protocols specific to the emerging IoT field: Message Queue Telemetry Transport (MQTT) and Bluetooth Low Energy (BLE). Based on their capabilities and characteristics, these two protocols, in tandem, can be considered the starting point in the design of any IoT modern application. Starting from the fact that MQTT is based on the client (publisher and subscriber)/broker paradigm, within the system proposed in the article under consideration, the MQTT broker runs on a single-board computer, Raspberry Pi3, and operates with two subscribers and one publisher. The MQTT publisher is an ESP32 microcontroller, which has at the same time the role of client in the context of the other used communication protocol - BLE, based on the client/server model. The first subscriber acts like an alarm for the system while the second subscriber is an WEB server developed based on the open-source NodeJS environment. Furthermore, the received data are stored in a database running on MongoDB and displayed from there in the web interface.**

***Index Terms*—IoT, MQTT, BLE, GAP, GATT, NodeJS, MongoDB, ESP32.**

I. INTRODUCTION

The Internet of Things (IoT), also known as the Internet of Everything, is a distinct system of Information and Communications Technology. It will bring massive changes in this area by integrating wireless communications, sensors and microcontrollers, together with digital processing techniques. According to the IoT paradigm, things have the ability to collect, process and transfer information within connected systems without any human intervention, thus requiring enhanced autonomous data processing skills [1]. The main purpose of this technology is to generate or gather information, in real-time, which can be further used for various purposes by the system in which technology is integrated. The Internet has been around for a long time, but it is used by people who design things for other people. On the contrary, the IoT represents an important and innovative technology that allows different devices to communicate and makes it possible to develop various applications which can be used as key elements in areas such as smart city, smart home, telehealth, self-driven car, etc. These connected devices, begin to share the information acquired from the environment using different sensors. The IoT devices can communicate with each other directly or through a cloud to which they are connected via the Internet.

The most important features of the Internet of Things are: distributivity, interoperability, scalability and security [2].

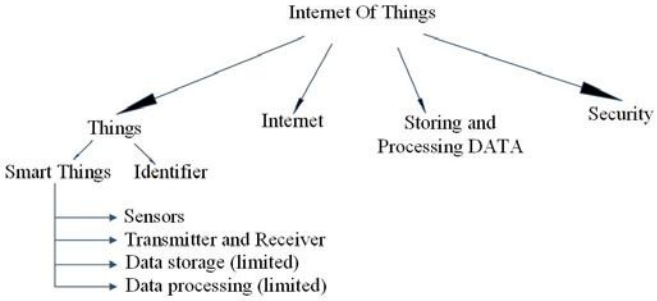

Figure 1. IoT Structure

In addition to fundamental IoT components, as shown in Fig. 1, IoT systems also consist of other essential elements as: data storage, data processing and security. Information storage and processing can be performed both at the network and local levels. The enhanced IoT systems involve smart objects that should have the ability to process the acquired data, by themselves, and to make decisions based on the information achieved. For this, to be possible, there must be a built-in data processing module that can process and analyze sensor data and make smart decisions. Thus, it can be noticed that a lot of data will be generated, stored and processed continuously.

This paper proposes a practical solution, for a communication system, based on MQTT protocols that fulfill the requirements imposed by the IoT technology. In our system, the publisher is represented by a client-server system that runs on the ESP32 microcontroller and is based on the BLE protocol. The subscriber, represented by a NodeJS web server, has the role of connecting the IoT system and the user. On the other hand, the server saves the data received through the broker in a database developed on MongoDB. Considering the implementation cost, for the transfer of data between Publisher-Broker and Broker-Subscriber, we considered the Wi-Fi technology at the expense of LoRa (Long Range) technology.

The paper is organized as follows: the MQTT and BLE protocols are introduced in Section II. The proposed practical IoT application is briefly presented in Sections III and IV with focus on database solutions and the communication protocols involved. Some conclusions and future works are drawn in Section V.

I. NARITA is with the Communications and Information Technology Department, Military Technical Academy 'Ferdinand I', Bucharest. Romania. (e-mail: ionut.narita@mta.ro).

F. POPESCU is with the Communications and Information Technology Department, Military Technical Academy 'Ferdinand I', Bucharest. Romania. (e-mail: florin.popescu@mta.ro).

C. DESPINA-STOIAN is with the Communications and Information Technology Department, Military Technical Academy 'Ferdinand I', Bucharest. Romania. (e-mail: cristina.despina@mta.ro).

II. IoT Communication Protocols

To create an MQTT IoT communication system, with the particularity of carrying short messages, we need to structure the network as a Publisher - Broker – Subscriber model. In this configuration, the publisher can be a terminal device that communicates with multiple sensors through an IoT protocol and transmits the information obtained to the Broker. The subscriber is associated with a client who wants to receive one or more types of data from the Broker and the Broker is the entity that ensures the communication between the publisher and the subscriber. Multiple publishers and subscribers may be connected to the broker and multiple subscribers may receive data from one publisher or a subscriber may receive data from a single publisher. Some of the most common wireless protocols that can underpin communication within these MQTT systems, especially at the edge of a network, are BLE and LoRa. In the implementation presented in this paper, we considered only one of them, respectively the BLE protocol which is detailed below.

A. BLE Protocol

BLE is a low-power wireless technology developed and used by various applications, including those defined by IoT systems. Considering the IoT's popularity, BLE technology is expected to be incorporated into billions of devices over the next few years and has been the area of research in recent years for various types of applications. Compared to other wireless technologies, such as Wi-Fi, the data rate of BLE technology is relatively low [3], but it is sufficient for most IoT applications developed with sensors on the front end of the network.

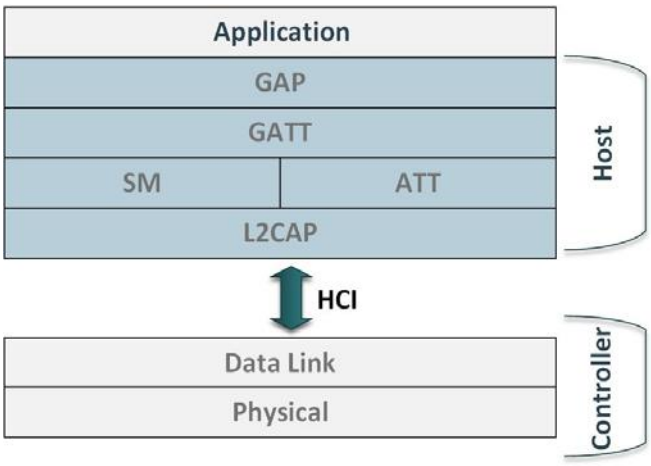

Figure 2. BLE Architecture

The BLE protocol stack consists of three main levels: controller, host, and application that are divided into sublevels as shown in Fig. 2. The controller level includes the physical layer and the data link layer. The host level includes the top-level functionality: Logical Link Control and Adaptation Protocol (L2CAP), Attribute Protocol (ATT), Generic Attribute Profile (GATT), Security Manager (SM), and Generic Access Profile (GAP). The host-controller communication is generically standardized as the Host Controller Interface (HCI) [4].

The main role of the L2CAP level is to multiplex, perform segmentation and reassemble the packages for the lower levels. ATT defines the data communication between two devices that act as a server and a client. SM which ensures a secure connection between client and server is responsible for pairing devices, authenticating and distributing keys. The data to be transmitted by one BLE node and received by another BLE node is stored in a database called the GATT server. The data on the GATT server can be transferred to another database, which is embedded in a node, this time of the GATT client type. From the MQTT perspective, in this approach, the GATT role is totally independent of the device role. Furthermore, the GATT server or GATT client can be placed anywhere, either at the broker level or at the publisher/subscriber level, depending on the requirements imposed by the application.

The GAP level [5] specifies the devices' role, operational modes and procedures for detecting BLE devices within range. In addition, it ensures services, connection management and security. The BLE GAP can impose four roles characterized by specific requirements for the basic controller: advertiser, scanner, central and peripheral. While advertising, scanning and initiating are states of BLE Link Layer used in the discovery process to find BLE devices.

The device that operates in the advertising mode is called the broadcaster, while the BLE device sets in scan modes are called observer. To discover your neighbors and receive advertisements, the scanner wakes up periodically. The functions of the scanner and the initiator (central) are the same, except that the scanner only intends to discover the advertiser, but the initiator may request a connection with the advertiser (peripheral) after receiving an advertising message. Connections ensure that the application-level data are transmitted reliably and robustly. For this, the BLE connections involve cyclic redundancy check (CRC), acknowledgment and retransmission mechanism that guarantee the correct delivery of data. Once a connection is established, the advertiser becomes a slave and the initiator becomes a master.

B. MQTT Protocol

The communication system is based also on the MQTT protocol. Suitable for applications developed on limited resource devices, the MQTT is a simple network protocol that provides communication through networks characterized by low bandwidth and high latency. It is designed to send short data messages that are specific to IoT applications. MQTT consists of two messages sets over a connection named “Publish" and "Subscribe" [6]. The message is identified in advance by the subject (topic) and recorded by the subscription message. Being developed for limited resource devices, the MQTT has specifications that are very easy to understand and apply [7].

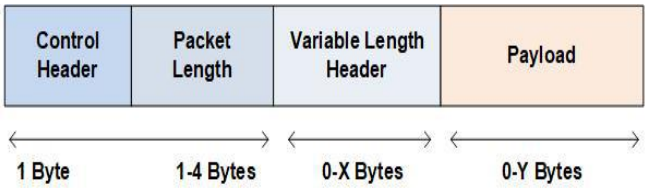

Figure 3. MQTT packet structure

The main MQTT type of messages are: CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE and

 Digital Object Identifier 10.32754/JMT.2021.2.08

DISCONNECT. Considering the limited number of message types, the MQTT packets involve a very small mandatory header of 2 bytes. Moreover, according to the standard header structure illustrated in Fig. 3, the payload is optional and its length depends on the packet type [8].

III. DATABASE SOLUTION FOR THE IOT APPLICATION

The data access level of the underground parking monitoring application is developed into the NodeJS [9] runtime environment, an open-source solution that stands out from other options by its asynchronous programming performance and the ability to run JavaScript outside of a browser. The NodeJS server does not create a new execution process for each request. In order to increase processing speed, it provides a set of asynchronous primitives that prevent difficulties by managing requests, significantly reducing the number of blockages, and eliminating the errors caused by the execution process. The following are just a few important criteria that make Node.js the first choice for software architects [10]: all Node.js library APIs are asynchronous event-driven, without blocking, very fast in executing the code, single-threaded, but highly scalable and no buffering, Node.js applications never store data.

Because NodeJS runs JavaScript code, it creates a JavaScript ecosystem in the proposed project and facilitates the development of a web application by consistency and unity, both in terms of user interface and server, without the need to switch from one programming language to another.

Once installed, NodeJS provides access to the NPM registry, which contains thousands of reusable packages. With the NPM package, Express JS is added, a framework responsible for managing HTTP requests and configuring connection settings.

The connection between the user interface and the data access level is maintained by accessing HTTP routes. Routes from the diagnostic application are requests to the Mongo database.

In Node JS is not required to have a certain structure of files. The programmer is free to manage the server architecture. The Model - View - Controller (MVC) architecture is considered for the underground parking monitoring application. The model is the part of the application that will deal with the functionality of the database, the view represents displaying data in a web interface, in this case, the requested data in the form of a JSON response and the controller that calls the model to receive data from Mongo and process it before use.

Mongo DB [11] is a NoSQL database that uses JSON documents as a storage method, which facilitates the development of the diagnostic application. The general rule that data that is accessed together must be stored together determines the optimality of queries.

NoSQL databases are differentiated from relational databases by storing data in the form of JSON documents, key-value pairs, and graphs, tables that contain only rows or columns. Depending on the requirements, a NoSQL database can be easily adapted with a flexible layout. They have a horizontal scaling, adding more servers/nodes to a system to cope with resources, and also vertical scaling means increasing the computing power of the existing system. Queries can achieve better performances because data sets do not need to be merged to meet demand, and data is stored in a default optimized form.

Moreover, the database selected for the proposed project has both a Cloud and an On-Premises solution on the personal server. The Cloud Mongo DB Atlas solution is used because it has noticeable advantages, being removed the maintenance part of the database, the attention of programmers turning to product development, this being desirable in an environment where people do not want to be forced to interact with the system. In general, Cloud Computing offers a wide range of services by renting resources, such as storage space and computing power. He is responsible for the physical equipment that performs the customer's requirements.

Using the Mongo DB Atlas interface in the browser, a cluster is created that contains the database for the underground parking monitoring application. Choose the Starter Cluster package, with the cloud provider Azure and a region for the data center.

Database queries are performed using an MDG called Mongoose that contains the necessary JavaScript functions to communicate with the database.

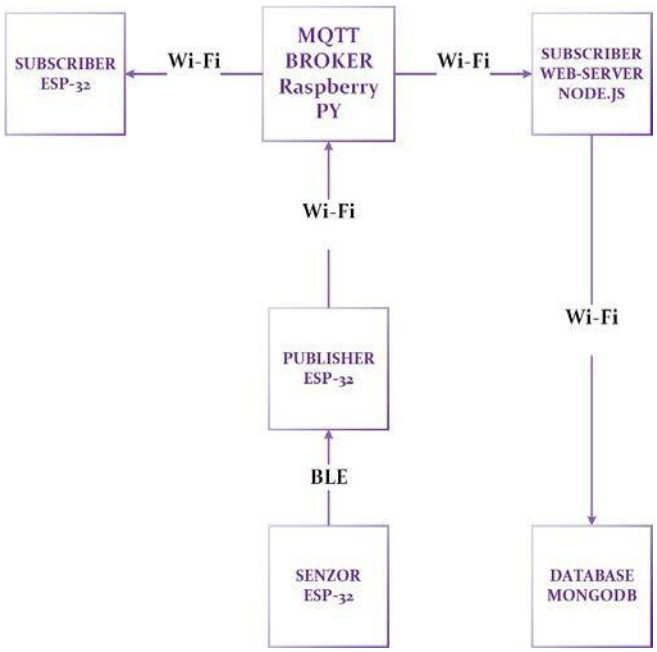

Figure 4. IoT communication system diagram

IV. PRACTICAL IMPLEMENTATION OF THE PROPOSED APPLICATION

The implemented blocks are the basis for the hardware and software implementation of an IoT communications system. Each block is configured to be part of a communications network that is based on the Mosquitto [12] Publisher - Broker - Subscriber model. Also, what makes this system to be considered an IoT communications system are the specific protocols for the Internet of Things, used to perform data transfer, MQTT, and BLE. The diagram of the proposed IoT system is presented in Fig. 4 with focus on the individual blocks and the communication protocols involved.

The MQTT communication system contains three components with specific roles and objectives. The first one, the publisher is the one who generates and sends data to the

MQTT broker. The second one, the broker is like a server that collects messages from every device that publishes information, saves data and distributes messages to the right subscribers. The third one, the subscriber is a component that subscribes to a certain type of message from the publisher.

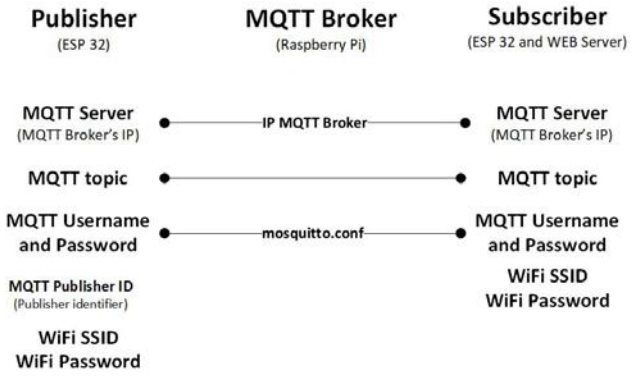

Figure 5. Configuration of the communication system parts

If we want to send data via MQTT, we need to define some variables and settings that link the publisher, broker and subscriber. Fig. 5 provides an overview of how the settings are defined and to which parts these settings are assigned.

The most important parameters used for publisher and subscriber configuration are summarized in Table I.

TABLE I. PUBLISHER AND SUBSCRIBER CONFIGURATION PARAMETERS

Variable	Settings
Server	172.20.10.6
Topic	Parking / Section D / temperature
User Name	XXXX
Password	XXXX
MQTT ID client	XXXX
Wi-Fi Network Parameters	Local Network SSID and Password

Firstly, the IP of the MQTT broker is the same IP address that the Raspberry Pi [13] has on the network. This IP address is a variable in both the publisher's and the subscriber's script, as both must connect to the broker. The second setting is the MQTT topic which is defined by the publisher. The data that is provided within a topic depends on the sensors used by the publisher. Since the goal is to monitor an underground car park and each topic which is built as a folder needs to be unique, the following topics can be obtained:

- Parking / Section D / temperature,
- Parking / Section D / CO2 level.

The advantage of using multiple topics is that one subscriber can be linked to all the topics that are part of section D, and a second subscriber can only be interested in the temperature inside the parking. The MQTT username and password are defined in the MQTT broker configuration file "*mosquitto.conf*". This configuration must be exactly the same in both the publisher's and the subscriber's script to gain access to the MQTT broker. Only the Publisher receives an MQTT client ID, which is identified within the network by this ID. The ID prevents the broker from receiving data from unknown publishers and also can identify the publisher who is sending this data. This ID can be associated with the customer's name. The last setting is also only for the publisher, SSID, and password of the Wi-Fi network to which it belongs. Without this information, the microcontroller cannot send data over the local wireless network.

In the current configuration, shown in Fig. 4, the sensor data are transmitted from the BLE server embedded in the "SENSOR ESP32" microcontroller to the "PUBLISHER ESP32", which forward data to Raspberry Pi broker, through a Wi-Fi channel. The data sent to the broker is taken over by two subscribers, one is a web server running in node.js and the other is an ESP32 microcontroller that acts as an alarm system.

The publishers presented in Fig. 4 are built on two ESP32 microcontrollers; one has connected a BMP280 [14] sensor as shown in Fig.6 and transmits data acquired via BLE to the next ESP32, which communicates Wi-Fi with the broker, with a communication path much longer in this way. The microcontroller that communicates with the broker can receive data from several sensors, thus forming a network node that makes the transition from data transmission via BLE to data transmission via Wi-Fi.

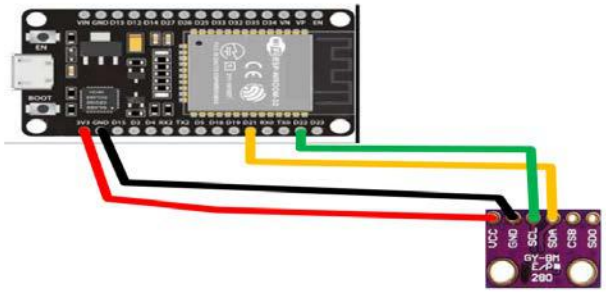

Figure 6. Publisher [Esp32 – BMP280 connections]

The data is transmitted from the server to the client via the BLE notifications. One example of such a notification is presented in Fig. 7. BLE server is designed to send notifications at a certain time to the one client who establishes the connection with it. Notifications are created as a buffer to store and read data from the sensor.

The read information is converted to char data before it is buffered.

```
19:44:57.068 -> Waiting a client connection to notify...
19:44:57.252 -> Temperature: 27.50 *C
```

Figure 7. Testing server connection with the client for BLE data transmission

To transfer the data, the publisher is connected with the broker through the private Wi-Fi network. There are software libraries, used for connecting to the MQTT broker, for publishing data and for connecting to the BLE server in order to receive notifications that contain sensor data that will be forwarded via Wi-Fi.

The *notifyCallback* function, shown in Fig. 8, uses a pointer, as an input parameter, that indicates the memory addresses in which the data received through notifications is stored, also their length. In order to be able to transmit data via the MQTT protocol, data must be in string format.

Once the the BLE server submitted data has been received and read, the connection to the broker must be made in order for it to be published. The connection is established via the WiFi network and is made using the function *connect_MQTT()*.

 Digital Object Identifier 10.32754/JMT.2021.2.08

The script waits until the wireless connection is established and the microcontroller has assigned an IP that permits its identification on the network. After setting up Wi-Fi transmission, we connect to the MQTT broker with the client ID, MQTT username and MQTT password. The loop function starts with the execution of the MQTT connection function, previously discussed.

```
static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify)
{
String valoare;
  Serial.print("Notify callback for characteristic ");
  Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
  Serial.print("Data: ");
  Serial.println(length);
  Serial.print("Mesajul primit ");
 for (int i = 0; i < length; i++)
 {
  char a = char(pData[i]);
   valoare= String (valoare + a);
 }
  Serial.println();
  Serial.print(valoare);
```

Figure 8. The *notifyCallback* function

Because we do not want to send data multiple times in a short amount of time, we can disconnect the Wi-Fi link after the MQTT data is sent and reconnect after a set amount of time. This helps to efficiently manage resources and reduce the use of energy by devices. After establishing the MQTT connection, to send the data strings to the broker, we use the client.publish() function from the PubSubClient library [15].

If the message is not sent successfully, we try to re-establish the connection with the broker and try retransmission of the data. The data are published and received by the devices that are subscribed to the same topic as the one that publishes the information, in this case, to the topic parking/section D/temperature.

In our topology two MQTT subscribers are presented: a web server made using NODE JS respectively an ESP32 microcontroller which is used as an alarm. Once the data is transmitted successfully to the broker, both subscribers have access to it. The first subscriber, NodeJS, takes the information and storage in a database then its taken and displayed in a web browser. The second subscriber takes the information from the broker, compares it to a standard level and if that level is exceeded it triggers the alarm.

Figure 9. BLE client-server connection

To summarize, the practical application is a network based on the Mosquitto MQTT principle, in which we have three essential devices: broker, publisher and subscriber. The publisher is an ESP32 microcontroller that also serves as a BLE client. In the case of the BLE protocol, the communication is client-server and is initiated by the server. The server sends notifications via a service defined by a universally unique identifier (UUID), as shown in Fig. 9, while the client receives notifications by connecting to the server via that UUID.

```
19:38:09.620 -> Connected to MQTT Broker!
19:38:09.620 -> Forming a connection to 3c:61:05:16:ad:36
19:38:09.620 ->  - Created client
19:38:09.714 ->  - Connected to server
19:38:10.271 ->  - Found our service
19:38:10.271 ->  - Found our characteristic
19:38:10.271 -> We are now connected to the BLE Server.
19:38:14.691 -> Notify callback for characteristic beb5483e-
19:38:14.736 -> Print out the message that we received27.42
19:38:14.736 -> 27.42Temperature sent!
```

Figure 10. Publisher to broker connection and data sharing

Once the data is received, it is sent to the broker (in fig. 10), receiving a feature that makes it unique, so that it can be received by the subscriber. The broker runs on a Raspberry Pi 3 and is connected to the publisher and subscriber via an Wi-Fi connection. The subscriber is a local server, made in NodeJS and which is subscribed to the broker by the publisher topic. When the data reaches the subscriber, it puts the data in a database, which is also displayed in a WEB interface. There is also another subscriber used as an alarm. The alarm is triggered when the data provided by the publisher exceeds a previously-imposed threshold.

V. Conclusion

The Internet of Things is constantly evolving in terms of services and capabilities and is becoming more widespread in key sectors. This technology is designed to help people, making their lives easier by performing essential tasks without the need for their intervention. IoT can be perceived as an innovation of what was previously known as the Internet because the Internet was created to be used by humans, while the Internet of Things is created to be used by devices for the benefit of people.

To enhance the Internet of Things feasibility, new low-power technologies and protocols have been developed for data traffic within these networks. The protocols used in IoT are constantly evolving to meet future requirements, with billions expected to be part of the network. The principles on which IoT protocols work are low power consumption, longer data propagation distances, and less data volume compared to non-IoT technologies.

Within the project, BLE and MQTT protocols were studied, their mode of operation, their advantages and disadvantages. Based on the theoretical notions and the practical aspects in which they can be used, the practical part, achieved as an ultimate goal, is an IoT communication system based on the BLE and MQTT protocols.

During the project, we encountered a few problems. One of these problems is the connection of the Apache WEB server to the MQTT broker, because it did not have a well-defined Mosquitto library. Thus, to fix this problem, we had to configure the server in NodeJS, proving to be a much better alternative for IoT systems.

Another problem encountered was the memory of the publisher who had to fulfill the role of BLE client in this application, thus exceeding the available memory. This problem has been solved by changing the memory partitioning scheme via Arduino IDE, allocating more memory for data and less for SPIFFS (Serial Peripheral

Interface Flash File System).

Also, another major problem was the data collision because we initially wanted to transmit data every 5 seconds, which is not possible, because after each transmission the publisher is planning to disconnect from the broker and Wi-Fi, for low power consumption. We managed to fix this problem by increasing the data transmission interval and we got an average transmission time of 14 seconds.

For the future development of this application, the use of Bluetooth mesh technology is considered, which can transmit data through several intermediate devices, thus making possible a much better communication inside buildings and over a much greater distance. It also aims to implement a LoRa network that interconnects the Publisher - Broker - Subscriber system.

REFERENCES

[1] P. Waher, “Learning Internet of Things”, Pack Publishing, 2015.
[2] M. R. Abdmeziem, D. Tandjaoui, R. Imed, “Architecting the Internet of Things: State of the Art” Robots and Sensor Clouds, Special edition in the “Studies in Systems, Decision and Control” Springer, 2015.
[3] C. Gomez, I. Demirkol, J. Paradells, Modeling the maximum throughput of bluetooth low energy in an error-prone link, IEEE Communications Letters, vol. 15, 2011, pag. 1187–1189.
[4] F. J. Dian, A. Yousefi and S. Lim, "A practical study on Bluetooth Low Energy (BLE) throughput," 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2018, pp. 768-771, doi: 10.1109/IEMCON.2018.8614763.
[5] https://www.bluetooth.com/specifications/specs/gatt-specification-supplement-4/
[6] https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-generic-access-profile-gap/
[7] MQTT-Topics Management System for Sharing of Open Data - Nasi Tantitharanukul, Kitisak Osathanunkul, Kittikorn Hantrakul, Part Pramokchon, and Paween Khoenkaw [2017 IEEE].
[8] MQTT Version 3.1.1(OASIS Standard), Andrew Banks and Rahul Gupta, 29 october 2014, available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.
[9] http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https
[10] https://nodejs.org/
[11] https://www.mongodb.com/
[12] https://mosquitto.org/
[13] https://www.raspberrypi.org/
[14] https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bmp280-barometric-pressure-plus-temperature-sensor-breakout.pdf
[15] https://www.arduino.cc/reference/en/libraries/pubsubclient/

 Digital Object Identifier 10.32754/JMT.2021.2.08