Development and Evaluation of Smart Home IoT Systems applied to HVAC Monitoring and Control

Ruben E. Figueiredo[1,*], Aníbal A. Alves[2], Vitor Monteiro[1], J. G. Pinto[1], João L. Afonso[1], and José A. Afonso[2]

[1] ALGORITMI Research Centre, University of Minho, Guimarães, Portugal
[2] CMEMS-UMinho Center, University of Minho, Guimarães, Portugal

Abstract

This work describes the development and evaluation of two smart home-based Internet of Things (IoT) systems applied to HVAC (Heating, Ventilation and Air Conditioning) monitoring and control, including parameters such as temperature, humidity, air quality, smoke detection, and human presence. These systems are based on a flexible hybrid wireless network architecture combining Bluetooth Low Energy (BLE) and IEEE 802.11/Wi-Fi, in order to adapt to the requirements of different types of sensor and actuator devices. The original implemented network is based on Cypress PSoC 4 BLE boards and HyperText Transfer Protocol (HTTP), whereas the new network uses ESP32 boards and includes Message Queue Telemetry Transport (MQTT), a lightweight messaging protocol suitable for IoT devices which provides additional quality of service (QoS) mechanisms to guarantee the delivery of messages. A smart temperature control system was implemented in the BLE/Wi-Fi gateway (Raspberry Pi) to keep the room temperature inside a user-defined range. An online database was also developed using the Amazon Web Services (AWS) cloud platform, allowing the users to access the HVAC data and control the system parameters, through the Internet, using a mobile app developed for Android devices. Experimental tests were performed to validate the functionalities and performance of the developed systems. The obtained results demonstrate that the new network provides lower delay values compared with the original implementation.

Keywords: Internet of Things, Smart Home, HVAC, Bluetooth Low Energy, Wi-Fi, MQTT, Wireless Sensor Networks.

Received on 05 September 2020, accepted on 17 November 2020, published on 19 November 2020

doi: 10.4108/eai.19-11-2020.167205

[*]Corresponding author. Email: ruben.figueiredo@algoritmi.uminho.pt

1. Introduction

Over the last few decades, technological advances enabled a large part of the world's population to have access to the Internet, and this access is mainly being done through mobile devices, using either cellular data networks or Wi-Fi. This trend, coupled with the increasing incorporation of sensor devices in a variety of equipment, opens a wide range of opportunities for the growing market of Internet of Things (IoT) applications, in areas such as transportation, healthcare, agriculture, industrial automation, smart home, among others [1].

The IoT enables physical objects to interact with the surrounding environment without requiring human intervention and to communicate with each other to share information and to coordinate decisions. The IoT allows connecting billions of objects through the Internet, so there is the need to define a layered architecture to handle the complexity associated with the different required tasks. In this sense, there has been an increasing number of proposed architectures, but there is not a consensual reference model yet. In [2], the authors present a five-layer model, where the first layer, Object, represents the physical sensors and actuators of the IoT that aim to collect and process information. The second layer, Object Abstraction, represents how the data is transferred from the physical objects. The third layer, Service Management, pairs a service with its requester, based on addresses and names. The fourth layer, Application, is responsible for providing high-quality smart services to meet customer's needs. Finally, the fifth layer, Business, defines the steps to build a business model based on the developed IoT system.

In order to maximize the lifetime of battery-operated sensor devices, it is desirable the use of low-power wireless sensor networks (WSN) technologies, such as Bluetooth Low Energy (BLE) [3] or IEEE 802.15.4/ZigBee [4]. WSNs enable new applications but require non-conventional paradigms for protocol design [5]. With characteristics such as low cost, low energy consumption [6], low latency and high reliability, as well a native hardware and software support provided by most current mobile devices, BLE takes a leading position for the implementation of IoT sensor devices over ZigBee in many areas of application [7][8]. However, since both BLE and ZigBee devices do not implement the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol stack, they require the introduction of a gateway device into the system to allow communication with other IoT devices, such as an IoT server or a mobile client [9].

To store the data collected by the sensor devices, a database is required. The successful implementation of an IoT system requires service provision with ubiquity, reliability, high-performance, efficiency and scalability. A way to achieve all of these goals is merging the IoT and the cloud computing concepts, as suggested in [10].

Concerning related work, in [11], the authors presented a networking solution for connecting BLE devices with the IoT, enabling end-to-end IP connectivity to the BLE devices in an efficient manner, especially in the aspects that are most critical for IoT devices: energy consumption and memory footprint of the implementation. In [9], the authors proposed a smartphone-based IoT gateway implemented as a software service that provides universal and ubiquitous Internet access to BLE connected IoT devices. This approach uses the smartphone both as an IPv6 router for less resource-constrained endpoints and as a BLE proxy, relaying profile data from the sensor device to the cloud.

This paper is an extended version of a previous conference paper [12]. The two smart home IoT systems are presented and evaluated in this paper using BLE to collect heating, ventilation and air conditioning (HVAC) data from sensor devices and send the information to an implemented BLE/Wi-Fi gateway, which also communicates with other local devices, such as actuators. Regarding data storage, the developed systems provide communication with a remote IoT server, in a cloud-based architecture, allowing the collected data to be accessible through the Internet. The new proposed system is based on the new BLE and Wi-Fi boards, introducing a more suitable messaging protocol at the application layer and also providing a local database as an offline alternative. A mobile app (client) was also developed in order to allow access to the data for the user. The developed systems are capable of smart temperature control on the desired room within a configurable temperature range.

The rest of this paper is organized as follows. Section 2 presents an overview of the proposed smart home IoT system architecture. Section 3 describes the implementation, in terms of hardware and software, of the original smart home network of the proposed IoT system, namely the BLE nodes and the gateway, while Section 4 provides an equivalent description for the new developed smart home network. Section 5 and Section 6 describe the development of the IoT cloud services and the IoT client (mobile app), respectively. Section 7 presents experimental results concerning the functional and non-functional aspects of the developed IoT system. Finally, Section 8 presents the conclusions.

2. System overview

The two smart home IoT systems developed in the context of this work share a common architecture composed by several components that exchange data with each other, as shown in Figure 1. Inside the smart home, the IoT devices communicate using a local hybrid BLE/Wi-Fi wireless network infrastructure, whose main components are the BLE sensor nodes, the BLE/Wi-Fi gateway, a wireless router (which provides connection to the Internet and acts as the local Wi-Fi access point) and actuator nodes. Besides the local components, the developed IoT system also includes an Android mobile app (client) and an Amazon Web Service (AWS) cloud server.

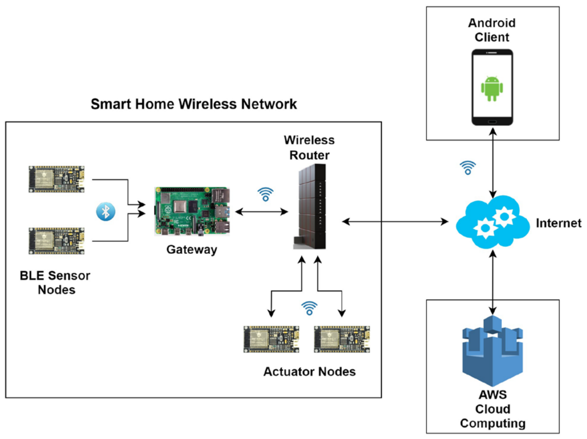

Figure 1. General architecture of the two developed smart home IoT systems.

The proposed architecture supports several sensor nodes and actuator nodes. Each BLE sensor node comprises two main components: a BLE device and a sensor, which may send data to the BLE device using an analog-to-digital converter (ADC) or a digital interface, such as Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI) or Inter-Integrated Circuit (I2C). Likewise, each actuator node is composed by a wireless device (either Wi-Fi or BLE) attached to an actuator.

The HVAC monitoring and control system built on top of the smart home IoT system works under the control of the local gateway even in case of failure of the Internet connection. When the Internet connection is available, the HVAC data collected by the BLE devices (sensor nodes) is also forwarded through the gateway, the Wi-Fi wireless router and the Internet infrastructure, until it reaches the cloud server, for storage. The Android client allows the user to access the data stored in the cloud server and send commands to configure and control the smart home devices (e.g., to define the minimum and maximum temperature for a room).

3. Original smart home network

The BLE devices used in the development and test of the system prototype were PSoC 4 BLE modules [13], from Cypress Semiconductor. Each BLE module was attached to a development board provided by the CY8CKIT-042-BLE-A kit, as shown in Figure 2. The BLE/Wi-Fi gateway was implemented using a Raspberry Pi 3 Model B [14], whereas the HVAC sensors and actuators were emulated using personal computers (PCs), which also acted as Wi-Fi devices for the actuator nodes. The development of each component of the IoT system is described in the next topics.

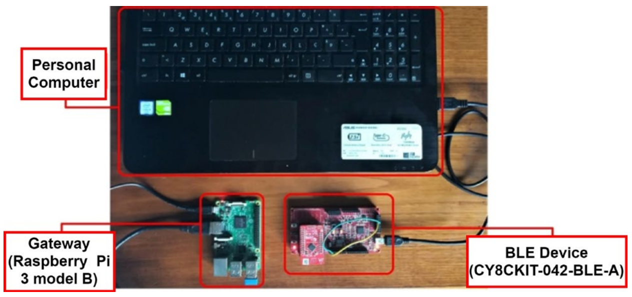
Personal Computer
Gateway (Raspberry Pi 3 model B)
BLE Device (CY8CKIT-042-BLE-A)

Figure 2. Main hardware components used in the development of the original smart home network.

3.1. BLE nodes design

This section describes the development of the BLE sensor nodes firmware. The BLE network is mainly responsible for collecting data, which in the context of the proposed application corresponds to HVAC parameters. In this sense, five representative sensors were considered: smoke detection, temperature, humidity, air quality and human presence detection. The data generated by these sensors were emulated using a PC-based Java application, which was developed using the IntelliJ IDEA IDE. The sensor data were transferred to the BLE modules using a serial data interface.

BLE devices have different roles at different layers of the Bluetooth protocol stack [15]. In this sense, the BLE modules were configured as slaves at the link layer, peripherals devices at the Generic Access Profile (GAP) layer and servers at the Generic Attribute Profile (GATT) layer, whereas the BLE/Wi-Fi gateway (Raspberry Pi) was configured as master, central device and client, respectively.

As aforementioned, the CY8CKIT-042-BLE-A development kit [13] was used for the implementation of the BLE slave/peripheral devices. Besides the PSoC 4 BLE module, this kit includes a development board (BLE pioneer), which allows programming and debugging the BLE module firmware through a PC. The C code for the BLE module microcontroller was developed using PSoC Creator 4.2 Integrated Development Environment (IDE). As referred before, all BLE modules act as peripheral devices, therefore, all of them include the same basic PSoC components.

In the PSoC Creator project IDE, the main component included in the design diagram of the sensor nodes is called BLE. This component is used to configure the BLE protocol parameters, such as advertising packets, connection interval, and the BLE notifications. It was necessary to create a GATT service and its characteristics. This component allows the use of predefined services, for example, a heart rate monitor or a proximity sensor, with its own characteristics; however, for this system, it was necessary to create new characteristics for each sensor value to be sent over BLE. Each sensor is connected to its respective BLE sensor node and has its own service. Of the five HVAC sensor values, one (smoke detection) is sent to the central device using BLE notifications, while the other four (temperature, humidity, air quality, and presence detection) are read by the central device (gateway) every 20 seconds (this time is configurable by the user in the app). To allow an automatic connection between the peripheral and central devices, it was necessary to include the Universally Unique Identifier (UUID) of the service in the advertisement packet, which was achieved in the “GAP Settings” tab of the BLE component. Two characteristics were created on the BLE component of each of the five sensor nodes: one characteristic represents the corresponding HVAC sensor value, whereas the other characteristic stores the identifier (ID) of the room where the sensor node was placed.

The second main component included in the design was a serial data interface, to collect the data from the sensors. In this prototype, we used the UART component provided by the PSoC Creator. For this component, it was only necessary to configure the same UART parameters as the Java application that was used to generate the HVAC data, such as the baud rate, which was set to 9600 bps.

A function called CustomEventHandler was used to detect and handle all events associated with the BLE stack. These events can be triggered by the central device when it connects or disconnects to the peripheral device or when the peripheral device announces its presence to the central device. It is also responsible for managing writing requests, made by the central device, to characteristics that have writing permission. On the developed system, the only characteristic with write permission was the room ID, which is configurable from the mobile app.

3.2. Gateway development

A Raspberry Pi 3 Model B was used to implement the BLE/Wi-Fi gateway and to act as the central device for the BLE network. The development was made in Python and using the Raspbian operating system. An external library called Pexpect was installed to allow the BLE communication with the peripheral devices. This library can generate processes related to certain applications, controlling them, and handle the response based on provided response patterns. On the developed gateway application software, it was used to automate the command "hcitool lescan" for monitoring BLE devices that are in an advertising state to central devices. JavaScript Object Notation (JSON) and Urllib2 libraries were also used, the former for converting data to JSON format and the latter for sending HyperText Transfer Protocol (HTTP) requests to store the collected data in the cloud. It was also necessary the installation of the BlueZ, an official Linux Bluetooth protocol stack, to handle the communication with BLE devices.

The developed BLE/Wi-Fi gateway provides bidirectional communication between the sensor nodes, the cloud server database, and the actuator nodes (which were implemented in Wi-Fi devices). For this purpose, the first task is to search and to connect to the desired BLE sensor nodes. Then, the central device needs to subscribe to notifications from the smoke detector. After that, the central device application starts to read the sensor values from the peripheral devices periodically and sends the data to the cloud database. Figure 3 shows a flowchart representing these tasks of the Raspberry Pi gateway application.

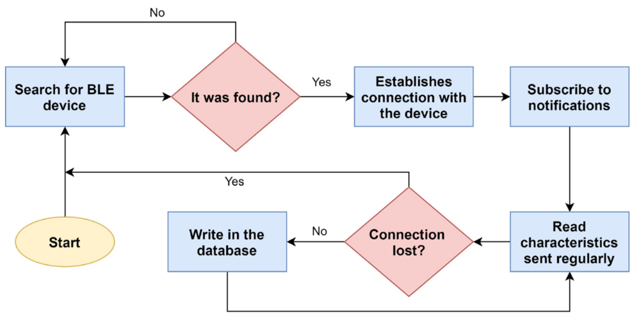
No
Search for BLE device
It was found?
Yes
Establishes connection with the device
Subscribe to notifications
Yes
Start
Write in the database
No
Connection lost?
Read characteristics sent regularly

Figure 3. Flowchart for the data collection and storage tasks of the Raspberry Pi application of the original smart home network.

4. New smart home network

The new IoT system that was developed has the same architecture as the original one, represented in Figure 1, but the communication between the IoT entities in the smart home relies on the Message Queue Telemetry Transport (MQTT) protocol, which is an Organization for the Advancement of Structured Information Standards (OASIS) and International Organization for Standardization (ISO) standard messaging protocol suitable for the Internet of Things.

In the developed prototype, ESP32 boards were used for both the sensor and actuator nodes, since they support both BLE and Wi-Fi, and the local gateway was implemented in a Raspberry Pi 4. The development of each component of the new smart home network is described in the following topics.

4.1. BLE nodes design

The FireBeetle ESP32 [16] development board was used for the implementation of the BLE sensor devices. The firmware was developed using the Arduino IDE, with the Arduino core for ESP32 developed by Espressif Systems [17]. In order to implement the BLE functionalities, the libraries made available by Neil Kolban [18] were imported.

Each BLE sensor node was designed to have its own service and two characteristics, as in the original smart home network, and the same characteristics properties (read, write, and notification) were attributed too.

To collect the data from attached sensors, the UART interface available in the ESP32 board was used. It was configured with the same UART parameters as the Python script that was used to generate the HVAC data, such as the baud rate, which was set to the same value used in the original network (9600 bps).

4.2. Gateway development

A Raspberry Pi 4 Model B [19] with the Raspbian as its main operating system was used to implement the gateway, replacing the Raspberry Pi 3 Model B used in the original smart home network. The gateway acts as the central brain of the system, running important processes and applications such as: automation and control algorithms; a BLE/Wi-Fi gateway; a local database; a HTTP server that provides access to the local database; a MQTT broker and others. In order to use the same IP address to access services and applications located on the Raspberry Pi, a static address to the network interface was configured and attributed.

MQTT was used in the development of the new network to handle the message exchange between the devices in the smart home. It requires minimal resources, so that it can be used with low-cost microcontrollers, as in ESP32 boards, has support for unreliable networks with limited bandwidth and uses an easy and simple publish/subscribe model that handles the message exchange, even if the IP addresses of the devices change along the time [20].

The MQTT protocol defines two types of network entities: the clients and a message broker. The clients are publishers if they generate data, such as sensor devices, and subscribers if they are interested in receiving data from particular topics, such as actuator devices. The broker is a server that manages the registrations for specific topics from publishers, receives all messages from the clients and routes them to the appropriate destinations.

MQTT offers a reliable message delivery with three quality service levels: 0 - at most once; 1 - at least once; and 2 - exactly once. In QoS level 0, the message is sent only once, and there is no checking if the message was delivered (fire and forget). QoS level 1 uses an acknowledgement message called PUBACK to confirm the delivery of the data message (acknowledged delivery). If the sender does not receive the acknowledgement, it retransmits the message. It is possible to receive a duplicate copy of the data message if the PUBACK message is lost. QoS level 2 uses a 4-way handshake mechanism to ensure that the data message is delivered exactly once (assured delivery) [21]. On the other hand, the exchange of these four messages can lead to higher end-to-end delays compared to the other QoS levels.

The MQTT broker was implemented through the installation of an open source and lightweight message broker called Eclipse Mosquitto [22]. Since MQTT clients cannot be installed in BLE devices, given that BLE do not use the Internet protocol suite, a script performing functions of a BLE/Wi-Fi gateway, as well as the MQTT client required for publishing the data generated by the BLE devices, were implemented in the Raspberry Pi, through the installation of the Eclipse Paho MQTT Python client library (paho-mqtt) [23], which implements the MQTT protocol, as well as bluepy [24], a Python module used to execute the BLE client to handle the wireless communication with the BLE devices.

For local data storage, it was used an open source database management system (DBMS) based on MySQL, called MariaDB [25], and installed the Apache HTTP server [26].

4.3. Wi-Fi nodes design

The Wi-Fi nodes are mainly used for actuator nodes, which do not have as many restrictions in terms of energy consumption. Therefore, considering the application scenario, the Wi-Fi nodes were implemented in the new smart home network too, but, unlike to the original network, these nodes were implemented using the same type of ESP32 boards used in the BLE sensor nodes, instead of being emulated using PCs.

The firmware was developed using the Arduino IDE, and the Joël Gähwiler MQTT library [27] was utilized to implement the MQTT client. The MQTT client acts as a publisher in the case of a sensor node, and acts as a subscriber in the case of an actuator node.

5. Cloud services development

This section describes the IoT services developed for the proposed system using the AWS cloud services platform, namely the database structure defined and the implemented functions. Instead of the traditional server-based approach, where the developer needs to handle the infrastructure management tasks, such as cluster provisioning, patching, operating system maintenance, and capacity provisioning, a serverless solution, which shifts these operational responsibilities to the AWS, was used. This solution is based on three individual services provided by the AWS: The Amazon Relational Database Service (RDS), the AWS Lambda, and the AWS API Gateway service. The choice of AWS over other cloud services providers was made based on an analysis of cost, performance and security [28].

5.1. RDS database

RDS is a free relational database service for new accounts during the first year, offering 750 hours per month. An alternative to this service is the Dynamo DB service, which is similar to RDS, but implements non-relational databases. The first step in the development of the database structure was to identify the data to be collected and to be shown to the user, which includes: (i) User data, representing the information provided when the user registers on the mobile app; (ii) Building data, containing the building address, name and ID; (iii) HVAC data, containing temperature, humidity, air quality, presence detection data, a timestamp and the room ID; (iv) Smoke detection data, containing the room ID and a timestamp; (v) Configuration data, including the maximum and minimum temperature values for the smart temperature control and other parameters.

For the implementation of the database on the AWS console, it was necessary to create an RDS instance, as well as making other configurations [29]. After that, the MySQL Workbench software [30] was used to develop all the tables and fields necessary to store the data. Even though smoke detection belongs to the HVAC parameters data, a separate MySQL table was created because this data was sent using BLE notifications, so it might have a different timestamp from the remaining parameters. It was also necessary to create inbound and outbound rules and apply them to the created instance in order to ensure access control to the data by other applications.

5.2. AWS Lambda

The AWS Lambda is a service that allows running code without provisioning or managing servers. This service executes the code when needed and scales automatically from a few requests per day to thousands per second. The free year offers 1 million requests to the created functions per month. The code can be run for any type of application or backend service with zero administration. AWS Lambda can be used in response to events, such as changes to data in the Amazon Dynamo DB or RDS tables, to run code in response to HTTP requests using the Amazon API Gateway or simply to invoke code using API calls made using AWS Software Development Kit (SDK).

For the proposed system, various functions were developed in NodeJS language. Each function is responsible for a functionality, as for example, getting the last HVAC parameters values from its correspondent table from the RDS database. The AWS console allows the development of the functions in two different ways: editing the code online or uploading zip files with the code and necessary packages inside. In this work, the second way was chosen. The AWS Lambda also allows testing the developed functions by providing a test event with a JSON body. All the data sent to and received from the AWS Lambda are in JSON format. Further configurations were necessary to give permissions to the functions created to access to the RDS database.

5.3. Amazon API gateway

The Amazon API Gateway is a service that makes easy to create, publish, maintain, monitor, and secure APIs at any scale. It can create Representational State Transfer (REST) and WebSocket APIs that allow applications to access data from backend services, such as AWS Lambda. In the proposed system, the API Gateway was used to connect the functions developed in the AWS Lambda to a REST API. When creating the REST API in the AWS console, it was necessary to create resources. In this system, a resource can represent the HVAC data or the buildings and is used to construct the path used on the HTTP requests methods. Each resource has been assigned to all the necessary methods, according to the needs by the different applications, such as GET, POST, DELETE or PUT. For the type of data to be received (JSON), it was necessary to configure each method and defining a body template for the GET methods in order to identify the parameters received by the HTTP requests. After creating the API, it was necessary to make it publicly accessible by creating a test stage. The Postman [31] software was used to test the created API.

6. Mobile app development

This section describes the implementation of the mobile app (IoT client). It was developed using the Android Studio IDE. This application communicates with the AWS database using the API Gateway service.

The Android app requires permissions to use the Internet. In order to accomplish with that, it is necessary to add dependencies to the AndroidManifest.xml file generated by the IDE when the application is created. The build.gradle file was also modified to allow the use of some required classes and layouts. Every layout implemented follows the Android guidelines by using the ConstraintLayout, which allows the application to run on any device, regardless its size. One of the most important classes used was the AsyncTask. This class allows that short asynchronous operations to run in the background, and it is usually used to perform network operations that do not require the download of much data. In the proposed system, it is used to do HTTP requests to the REST API.

The application allows the users to register or login using the classic email password combination. For registration, the user only needs his email, name, and password. After the login, the user is presented with a list of buildings that he/she has access and can eliminate or add new ones. A long click on a building allows the user to go to the building rooms and a list of rooms is presented to the user. The user can add new rooms by simply introducing the room name or delete those already created. When the user adds a new room, a table is automatically created that contains a default maximum and minimum temperature values for that room. Clicking on a room opens the information panel related to its HVAC parameter values. The user can see the most recent values collected or change the temperature interval and room ID.

7. Experimental results and discussion

This section presents the results of the experimental tests performed for the overall system and involves the evaluation of both functional features (data collection and presentation) and non-functional features (communication delay and reliability).

7.1. Data collection and presentation

The gateway is responsible for receiving the HVAC data from the BLE sensor nodes, process, and send it to the AWS database and/or the actuators and handle the smart temperature control process. The mobile app, on the other hand, is responsible for presenting the collected data to the user and allowing manual control of the system.

The application has a bottom navigation menu that makes easy to change between functionalities. The default choice of the bottom navigation menu is the HVAC screen (Figure 4), which shows the timestamp of the collected data, the temperature, humidity and air quality reading, as well as the state of the heating (on or off). The second option (Detectors) shows the data regarding the smoke and presence detectors and their corresponding timestamps. The last option (Configurations) allows the user to check and change the desired maximum and minimum temperature values and change the ID of the room where a sensor is located.

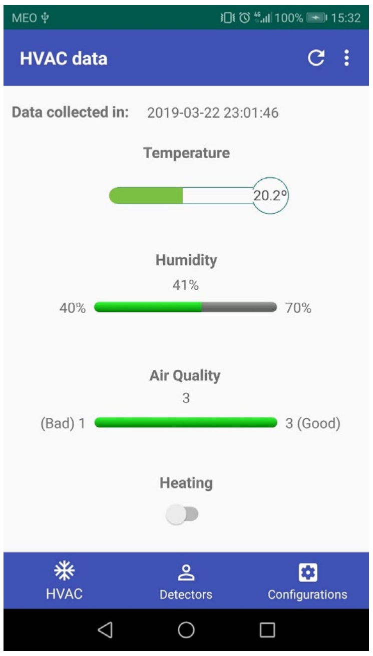

Figure 4. Example of values presented on the HVAC screen of the developed mobile app.

7.2. Communication delay and reliability

The communication delay, from the time instant that the sensor data is generated until the control information is delivered to the respective actuator, is an important parameter, since it affects the system performance, namely the response time of the control system. Despite in the proposed application the smart temperature control is not very demanding, since the room temperature changes slowly, a test setup was conceived and implemented in order to evaluate the performance associated to the temperature data sensing/actuation process, as shown in Figure 5. The total delay is the sum of several partial delays in the path through different devices, from the source to the destination, including data transmission times in the different data interfaces (UART, BLE, and Wi-Fi), as well as medium access delays and processing delays. The measured total delay corresponds to the time elapsed since the data is sent by the source (start time) until it is received in the destination (end time). The same device (a PC) was used as a source and destination in order to provide a common clock, which is necessary for a precise measurement of the delay.

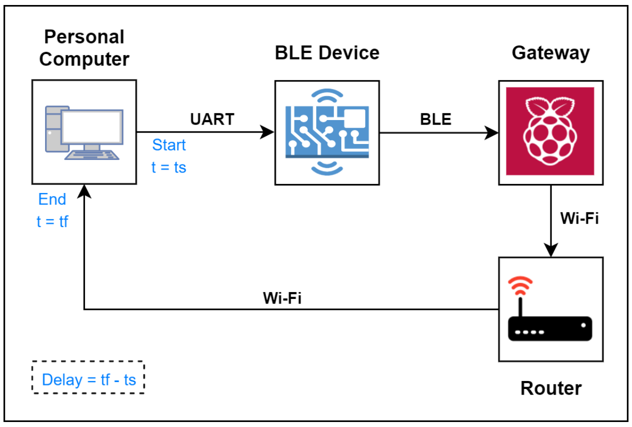

Figure 5. Method used to measure the communication delay for each packet.

In each test, 1000 data packets were generated at the source and the same amount was received at the destination; therefore, the communication reliability was 100%. Table 1 shows the main representative statistics for the delay measured in the performed tests: minimum, mean, maximum and standard deviation (SD).

An analysis of the delay distribution results from these tests shows that the original network has 96.8% of the delay samples in the range from 100 ms to 299 ms (Figure 6). On the other hand, the new network has 98.4% of the delay samples in the range from 20 ms to 99 ms with QoS 1 (Figure 7) and 95.1% of the delay samples in the range from 20 ms to 139 ms with QoS 2, (Figure 8). These results show that the new smart home network has better performance than the original one, and that, as expected, QoS 2 has slightly higher delay values than QoS 1.

Table 1. Main statistics concerning the measured communication delay.

Tests	Min. (ms)	Mean (ms)	Max. (ms)	SD (%)
Original	110	194	563	57.6
New, QoS 1	22	61	203	31.2
New, QoS 2	29	85	361	36.9

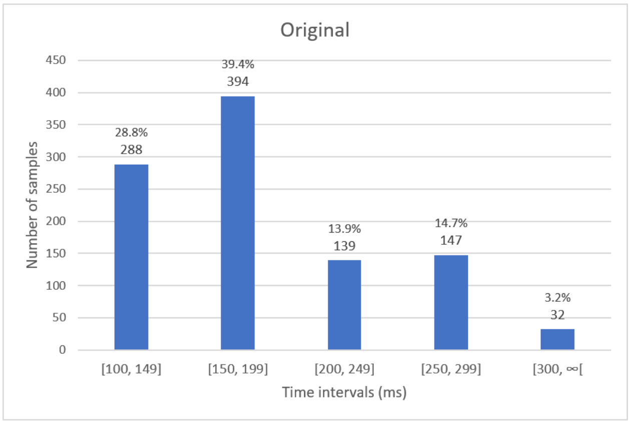

Figure 6. Delay distribution for the test performed with the original smart home network.

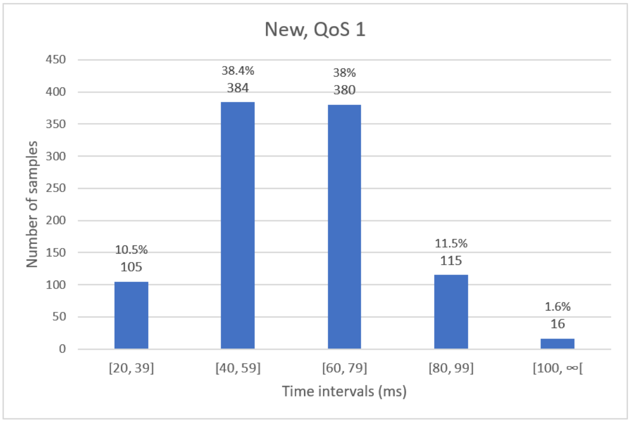

Figure 7. Delay distribution for the test performed with the new smart home network and MQTT QoS 1.

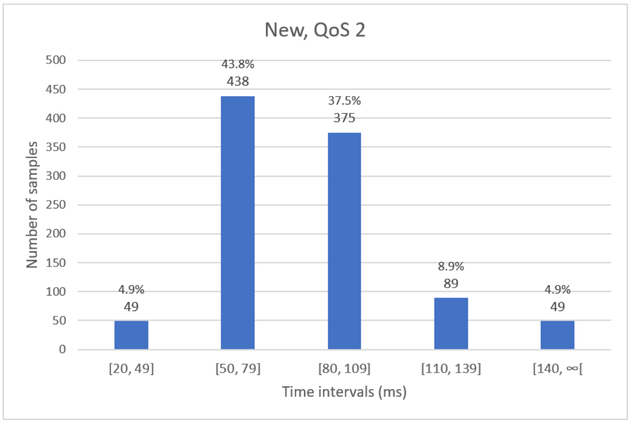

Figure 8. Delay distribution for the test performed with the new smart home network and MQTT QoS 2.

The maximum delay values observed, in all cases (563 ms for the original network, 203 ms with QoS 1, and 351 ms with QoS 2) are below the typical HVAC time constraints. Nevertheless, the new smart home network provides additional QoS mechanisms to guarantee the delivery of messages and presents lower time delay values when compared with the original network.

Regarding the MQTT QoS level to be used in the system, QoS 0 was not considered since it does not guarantee the delivery at the destination. Based on the obtained results, we advise the use of QoS 2 because, although the QoS 1 delays are lower, duplicate messages may occur and possibly affect the behaviour of the system, which means that the receiver implementation would require additional logic to detect duplicates.

8. Conclusion

This paper described the development and evaluation of two smart home IoT systems applied to HVAC monitoring and control using a mobile app. The proposed systems are based on a local hybrid BLE/Wi-Fi wireless network infrastructure and are composed by multiple data processing and communication components that work together to perform the desired functions.

The BLE/Wi-Fi gateway plays a central role in this system, with relevance to both the data communication and the processing algorithms of the HVAC application, such as in the case of the smart temperature control algorithm. An online database was also developed using the AWS cloud platform, in a serverless approach, and a mobile app (IoT client) was developed for the Android mobile operating system.

The developed systems were validated through experimental tests comprising the evaluation of their main functionalities, ranging from data collection at the BLE sensor nodes to the presentation at the mobile app, as well as the evaluation of its performance in the path between the sensors and actuators. The communication reliability was 100% for both versions, but the new smart home network presented better performance. Besides that, the new network introduces several new features, such as the implementation of Wi-Fi actuator nodes using the same type of wireless communication board (FireBeetle ESP32) used in the BLE sensor nodes, and the provision of a local database implemented using MariaDB as an alternative/ complement to the cloud RDS database.

Besides the considered application scenario, the proposed smart home IoT architecture and the associated message protocols may also be extended to use in a wide range of other application areas, such as lighting control and security, as well as in smart grids and microgrids.

Acknowledgements.
This work was supported by FCT national funds, under the national support to R&D units grant, through the reference project UIDB/04436/2020 and UIDP/04436/2020.

References

[1] Ullo, Silvia Liberata, and G R Sinha, "Advances in Smart Environment Monitoring Systems Using IoT and Sensors," Sensors (Basel, Switzerland), vol. 20, 11 3113, 31 May. 2020, doi:10.3390/s20113113.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” IEEE Communications Surveys & Tutorials, vol. 17, no. 4, 2015, pp. 2347-2376.

[3] J.A. Afonso, A.J.F. Maio, and R. Simoes, “Performance evaluation of Bluetooth Low Energy for high data rate body area networks,” Wireless Personal Communications, vol. 90, no. 1, Sept. 2016, pp. 121-141.

[4] P. Castro, J.L. Afonso, and J.A. Afonso, “A Low-Cost ZigBee-based Wireless Industrial Automation System,” 12th Portuguese Conference on Automatic Control, Guimaraes, Portugal, 2016.

[5] C. Buratti, A. Conti, D. Dardari, and R. Verdone, “An overview on wireless sensor networks technology and evolution,” Sensors, vol. 9, no. 9, Aug. 2009, pp. 6869-6896.

[6] S. Kamath and J. Lindh, “Measuring Bluetooth Low Energy power consumption,” Application Note AN092, Texas Instruments, 2012, pp. 1-24.

[7] M. Siekkinen, M. Hiienkari, J.K. Nurminen, and J. Nieminen, “How low energy is Bluetooth Low Energy? Comparative measurements with ZigBee/802.15.4,” IEEE WCNCW Wireless Communications and Networking Conference Workshops, Apr. 2012, pp. 232-237.

[8] P. Trelsmo, P. Di Marco, P. Skillermark, R. Chirikov and J. Ostman, "Evaluating IPv6 Connectivity for IEEE 802.15.4 and Bluetooth Low Energy," 2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), San Francisco, CA, 2017, pp. 1-6, doi: 10.1109/WCNCW.2017.7919088.

[9] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson and P. Dutta, “The Internet of Things Has a Gateway Problem,” 16th International Workshop on Mobile Computing Systems and Applications, 2015, pp. 27-32.

[10] A. Biswas and R. Giaffreda, “IoT and cloud convergence: Opportunities and challenges,” IEEE World Forum on Internet of Things (WF-IoT), Seoul, South Korea, 2014.

[11] J. Nieminem et al., “Networking Solutions for Connecting Bluetooth Low Energy Enabled Machines to the Internet of Things,” IEEE network, vol. 28, no. 3, 2014.

[12] Aníbal A. Alves, Vitor Monteiro, J. G. Pinto, J. L. Afonso and José A. Afonso, “Development of an Internet of Things System for Smart Home HVAC Monitoring and Control”, SESC (2019), Braga, Portugal.

[13] Cypress Semiconductor, “CY8CKIT-042-BLE-A Bluetooth Low Energy 4.2 Compliant Pioneer Kit.” [Online]. Available: https://www.cypress.com/documentation/development-kitsboards/cy8ckit-042-ble-bluetooth-low-energy-42-compliant-pioneer-kit.

[14] Raspberry Pi Foundation, “Raspberry Pi 3 Model B.” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[15] Bluetooth Special Interest Group. Specification of the Bluetooth System, Covered Core Package Version: 5.0, Kirkland, WA, USA, Dec. 2014.

[16] DFRobot, “Firebeetle ESP32.” [Online]. Available: https://wiki.dfrobot.com/FireBeetle_ESP32_IOT_Microcontroller(V3.0)__Supports_WiFi_&_Bluetooth__SKU__DFR0478.

[17] Espressif Systems, “Arduino core for the ESP32”. [Online]. Available: https://github.com/espressif/arduino-esp32.

[18] N. Kolban, “esp32-snippets/BLE C++ Guide”. [Online]. Available: https://github.com/nkolban/esp32-snippets/blob/master/Documentation/BLE C%2B%2B Guide.pdf.

[19] Raspberry Pi Foundation, “Raspberry Pi 4 Model B.” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.

[20] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh and R. Al-Hatmi, "Internet of Things: Survey and open issues of MQTT protocol," 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, 2017, pp. 1-6, doi: 10.1109/ICEMIS.2017.8273112.

[21] K. Grgić, I. Špeh and I. Heđi, "A web-based IoT solution for monitoring data using MQTT protocol," 2016 International Conference on Smart Systems and Technologies (SST), Osijek, 2016, pp. 249-253, doi: 10.1109/SST.2016.7765668.

[22] Eclipse Foundation, “Eclipse Mosquitto”. [Online]. Available: https://mosquitto.org/.

[23] Eclipse Paho, “MQTT and MQTT-SN software”. [Online]. Available: https://www.eclipse.org/paho/clients/python/.

[24] Harvey I., “Python interface to Bluetooth LE on Linux”. [Online]. Available: https://github.com/IanHarvey/bluepy.

[25] MariaDB Foundation, “About MariaDB - MariaDB.org”. [Online]. Available: https://mariadb.org/about/.

[26] Apache, “Apache HTTP server project”. [Online]. Available: https://httpd.apache.org/.

[27] Joël Gähwiler, “MQTT”. [Online]. Available: https://github.com/256dpi/arduino-mqtt.

[28] Amazon, “Amazon Web Services (AWS) - Cloud Computing Services.” [Online]. Available: https://aws.amazon.com/pt/.

[29] Amazon, “10-Minute Tutorials with Amazon Web Services (AWS).” [Online]. Available: https://aws.amazon.com/getting-started/tutorials/.

[30] Oracle Corporation, “MySQL: MySQL Workbench.” [Online]. Available: https://www.mysql.com/products/workbench/.

[31] Postman, Inc., “POSTMAN | API Development Environment.” [Online]. Available: https://www.getpostman.com/.