Development of an Internet of Things System for Smart Home HVAC Monitoring and Control

Aníbal A. Alves[1], Vitor Monteiro[2], J. G. Pinto[2], Joao L. Afonso[2], and Jose A. Afonso[1(✉)]

[1] CMEMS-UMinho Center, University of Minho, Guimarães, Portugal
jose.afonso@dei.uminho.pt
[2] ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

Abstract. This paper presents the development and test of an Internet of Things (IoT) system applied to the monitoring and control of an HVAC (Heating, Ventilation and Air Conditioning) system that includes parameters such as temperature, humidity, air quality, human presence and smoke detection. For this purpose, a hybrid wireless network combining Bluetooth Low Energy (BLE) and IEEE 802.11/Wi-Fi was implemented inside a house. An online database for the synchronization of the HVAC data, which was developed using the Amazon Web Services (AWS) cloud platform, allows the user to access the data and control the system parameters through the Internet using an Android mobile app. A smart temperature control system was also developed in the BLE/Wi-Fi gateway to keep the room temperature inside a user-defined range. The functionalities and performance of the proposed system were both validated through experimental tests.

Keywords: Internet of Things · Smart home · Bluetooth Low Energy · Wireless sensor networks

1 Introduction

Over the last few decades, technological advances have enabled a large part of the world's population to have access to the Internet, and this access is increasingly 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.

The IoT enables physical objects to interact with the surround 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 to 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

J. L. Afonso et al. (Eds.): SESC 2019, LNICST 315, pp. 197–208, 2020.
https://doi.org/10.1007/978-3-030-45694-8_15

[1], the authors present a five-layer model, where the first one, Object's layer, 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) [2] or IEEE 802.15.4/ZigBee [3]. WSNs enable new applications but require non-conventional paradigms for protocol design [4]. With characteristics such as low cost, low energy consumption [5], 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 [6]. However, since both BLE and ZigBee devices do not implement the TCP/IP (Transmission Control Protocol/Internet Protocol) 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 [7].

In order 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 [8].

Concerning related work, in [9], 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 aspect that are most critical for IoT devices: energy consumption and memory footprint of the implementation. In [7], 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.

The smart home IoT system presented in this paper uses 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 system provides communication with a remote IoT server, in a cloud-based architecture, allowing the collected data to be accessible through the Internet. A mobile app (client) was also developed in order to allow access to the data for the user. The developed system is capable of a smart temperature control on a desired room inside a configurable temperature range.

The rest of this paper is organized as follows. Section 2 presents an overview of the developed system architecture and components. Section 3 describes the development of the home network components, namely the BLE nodes and the gateway. Sections 4 and 5 describe the development of the IoT cloud services and the IoT client (mobile app), respectively. Section 6 presents experimental results concerning functional and non-functional aspects of the developed IoT system. Finally, Sect. 7 presents the conclusions.

2 System Overview

The developed IoT system is constituted by several components that exchange data with each other, as shown in Fig. 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 AWS (Amazon Web Service) cloud server.

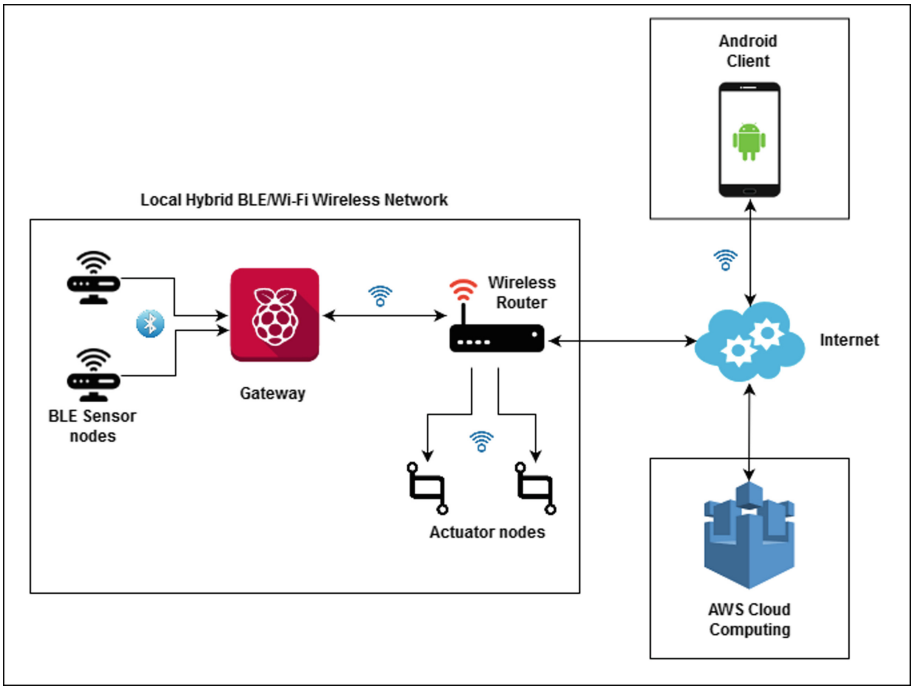

Fig. 1. Architecture of the developed IoT system.

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 UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface) or I2C (Inter-Integrated Circuit). Likewise, each actuator node is composed by a wireless device (either Wi-Fi or BLE) attached to an actuator.

The HVAC 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).

The BLE devices used in the development and test of the system prototype were PSoC 4 BLE modules [10], from Cypress Semiconductor. Each BLE module was attached to a development board provided by the CY8CKIT-042-BLE-A kit, as shown in Fig. 2. The BLE/Wi-Fi gateway was implemented using a Raspberry Pi 3 Model B [11], whereas the HVAC sensors and actuators were emulated using personal computers (PCs), which also acted as the actuator nodes' Wi-Fi devices. The development of each component of the IoT system is described in the next sections.

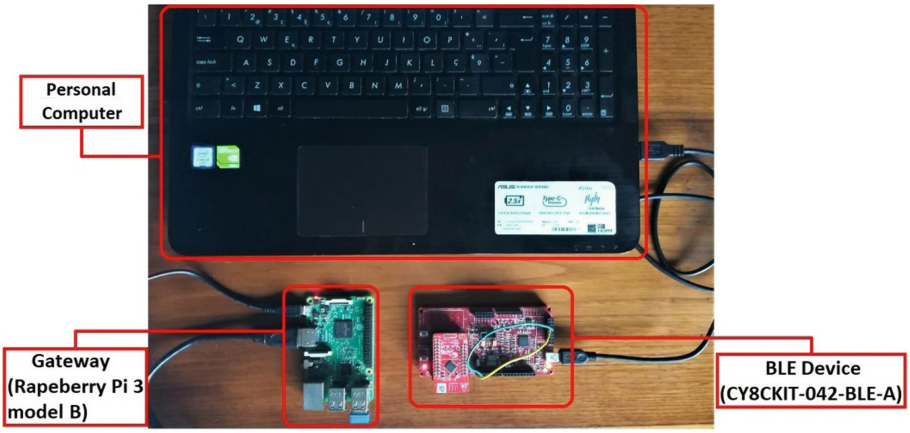

Fig. 2. Main hardware components used in the development of the IoT system.

3 BLE Network Development

This section describes the development of the firmware of the BLE sensor nodes. 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 was emulated using a PC-based Java application, which was developed using the IntelliJ IDEA IDE. The sensor data was transferred to the BLE modules using a serial data interface.

BLE devices have different roles at different layers of the Bluetooth protocol stack [12]. In this sense, the BLE modules were configured as slaves at the link layer, peripherals devices at the GAP (Generic Access Profile) layer and servers at the GATT (Generic Attribute Profile) layer, whereas the BLE/Wi-Fi gateway (Raspberry Pi) was configured as master, central device and client, respectively.

3.1 BLE Nodes Design

As referred before, the CY8CKIT-042-BLE-A development kit [10] 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 environment, 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) each 20 s (configurable). In order to allow the connection between the peripheral and central devices to be made automatically, 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 ID (identifier) 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 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. JSON (JavaScript Object Notation) and Urllib2 libraries were also used, the former for converting data to JSON format and the latter

for sending HTTP (HyperText Transfer Protocol) requests to store the collected data in the cloud. It was also necessary the installation of 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 as mains powered Wi-Fi devices). For this purpose, the first task is to search and 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 send the data to the cloud database. Figure 3 shows a flowchart representing these tasks of the Raspberry Pi gateway application.

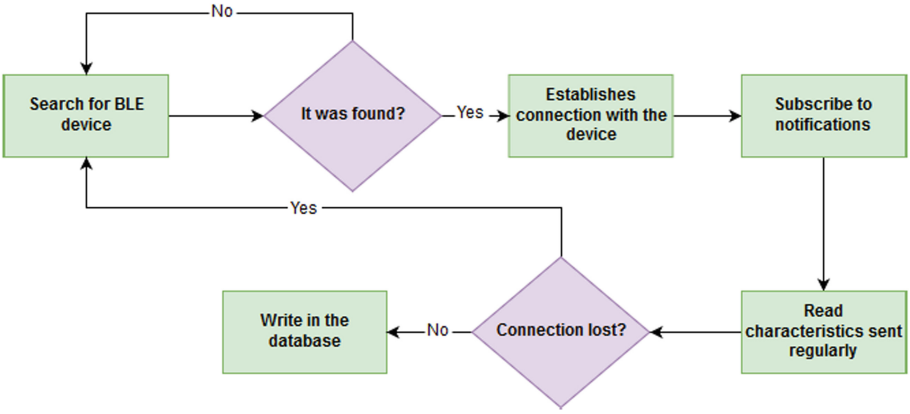

Fig. 3. Flowchart for the data collection and storage tasks of the Raspberry Pi application.

The Raspberry Pi application is also responsible for the smart control of the temperature, sending commands to turn on/off the actuator. Each time this application reads a temperature value and room ID from a BLE peripheral device, it checks if the temperature is inside the defined range for that room (stored in the gateway). If not, the application sends a command via Wi-Fi socket to the corresponding actuator, changing its state accordingly. This is possible because the gateway stores the IP addresses of the actuators.

4 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 [13].

4.1 RDS Database

RDS is a free relational database service for new accounts during the first year, offering 750 h per month. An alternative to this service is the Dynamo DB service, which 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 and 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 [14]. After that, the MySQL Workbench software [15] 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.

4.2 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 SDKs (Software Development Kit).

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 is in JSON format. Further configurations were necessary to give permissions to the functions created to access to the RDS database.

4.3 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 REST (Representational State Transfer) 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 build 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 the creation of the API, it was necessary to make it publicly accessible by creating a test stage. The Postman [16] software was used to test the created API.

5 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 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.

6 Experimental Results

This section presents results based on 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).

6.1 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 (Fig. 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.

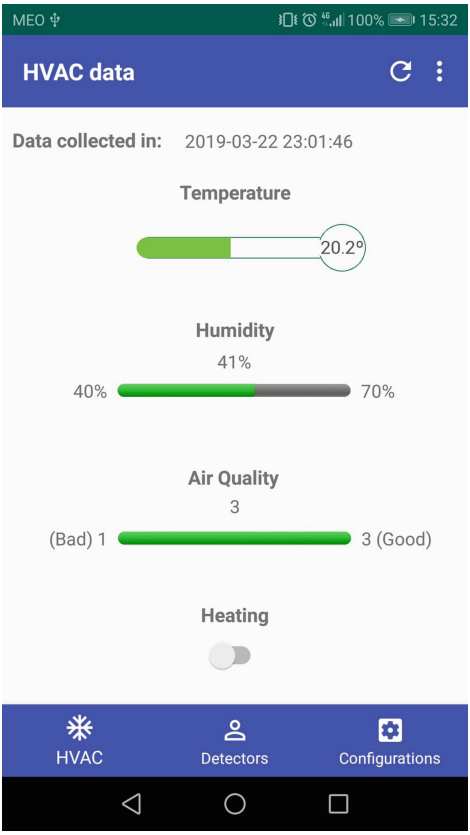

Fig. 4. Example of values presented on the HVAC screen of the developed mobile app.

6.2 Communication Delay and Reliability

The communication delay, from the moment that the sensor data is generated until the control information is delivered to the respective actuator, is an important parameter, since it affects the performance of the system, namely the response time of the smart temperature control system. Therefore, a test setup was conceived and implemented in order to evaluate the performance associated to the temperature data sensing/actuation process, as shown in Fig. 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 running a Java application) was used as source and destination in order to provide a common clock, which is necessary for the calculation of the delay.

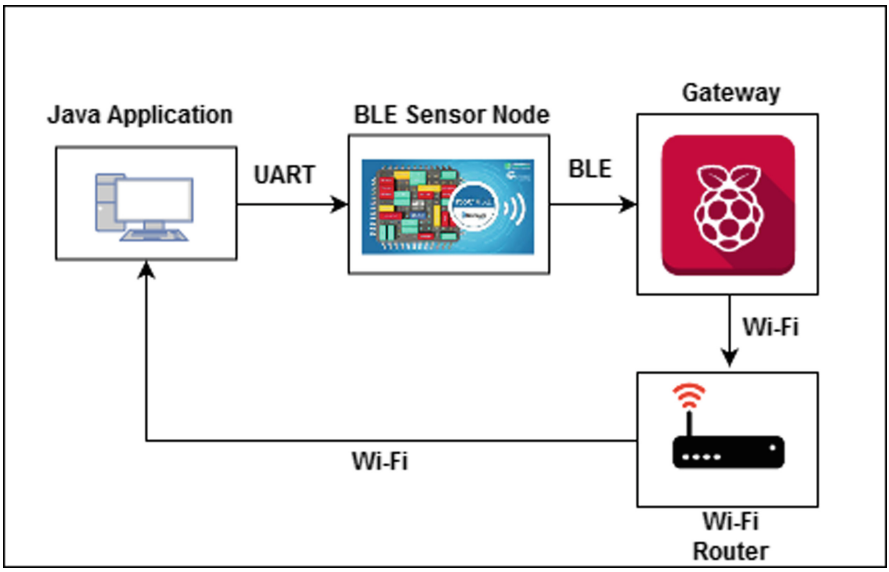

Fig. 5. Test structure for measuring the communication delay.

During the test, 1000 data packets were generated at the source and the same amount was received at the destination; therefore, the communication reliability was 100%. The test was replicated a second time, with similar results. Table 1 shows the main representative measures for the delay obtained in the two performed tests, minimum, maximum, mean and standard deviation (SD), where the maximum values are well below the typical HVAC deadline requirements. Figure 6 shows the distribution of the communication delay for the samples obtained during test 1, where it can be seen that 97% of the delay samples are in the range from 100 to 300 ms. These results are satisfactory, given the slow evolution of the HVAC parameters along the time.

Table 1. Main statistics concerning the measured communication delay.

Tests	Min. (ms)	Max. (ms)	Mean (ms)	SD (ms)
Test 1	110	563	194	57.6
Test 2	109	320	186	43.6

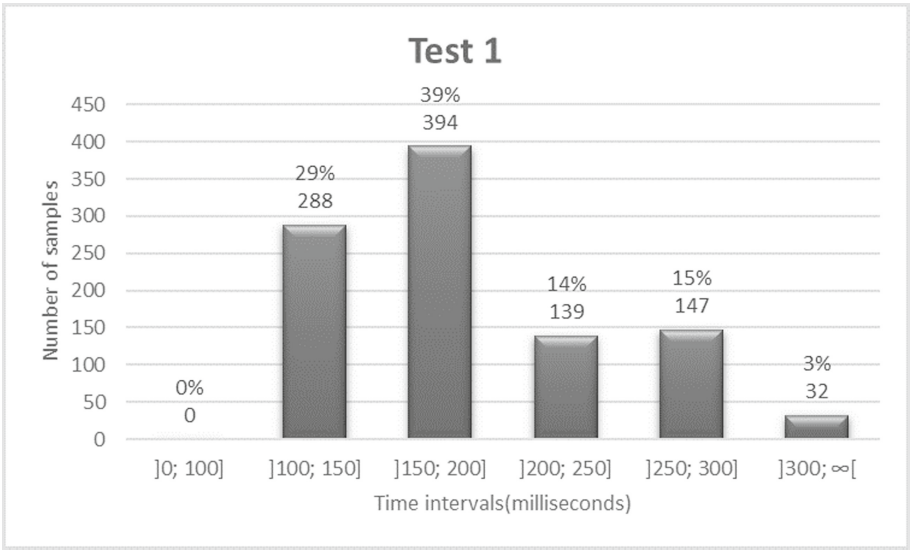

Fig. 6. Distribution of the communication delay for test 1.

7 Conclusions

This paper described the development of a smart IoT system that allows the user to monitor and control HVAC parameters in a smart home using a mobile app. The proposed system is 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 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 system was validated through experimental tests comprising the evaluation of its 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% and the obtained delay results are adequate, since the variation of the HVAC parameters along the time occurs in a much slower way.

Acknowledgments. This work is supported by FCT with the reference project UID/EEA/04436/2019.

References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutor. **17**(4), 2347–2376 (2015)
2. Afonso, J.A., Maio, A.J.F., Simoes, R.: Performance evaluation of Bluetooth Low Energy for high data rate body area networks. Wirel. Pers. Commun. **90**(1), 121–141 (2016). https://doi.org/10.1007/s11277-016-3335-4
3. Castro, P., Afonso, João L., Afonso, José A.: A low-cost ZigBee-based wireless industrial automation system. In: Garrido, P., Soares, F., Moreira, A.P. (eds.) CONTROLO 2016. LNEE, vol. 402, pp. 739–749. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-43671-5_62
4. Buratti, C., Conti, A., Dardari, D., Verdone, R.: An overview on wireless sensor networks technology and evolution. Sensors **9**(9), 6869–6896 (2009)
5. Kamath, S., Lindh, J.: Measuring Bluetooth Low Energy power consumption. In: Application Note AN092, Texas Instruments, pp. 1–24 (2012)
6. Siekkinen, M., Hiienkari, M., Nurminen, J.K., Nieminen, J.: How low energy is bluetooth low energy? comparative measurements with ZigBee/802.15.4. In: IEEE WCNCW Wireless Communications and Networking Conference Workshops, pp. 232–237, April 2012
7. Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., Dutta, P.: The internet of things has a gateway problem. In: 16th International Workshop on Mobile Computing Systems and Applications, pp. 27–32 (2015)
8. Biswas, A., Giaffreda, R.: IoT and cloud convergence: opportunities and challenges. In: IEEE World Forum on Internet of Things (WF-IoT), Seoul, South Korea (2014)
9. Nieminem, J., et al.: Networking solutions for connecting bluetooth low energy enabled machines to the internet of things. IEEE Netw. **28**(3), 83–90 (2014)
10. Cypress Semiconductor: CY8CKIT-042-BLE-A Bluetooth Low Energy 4.2 Compliant Pioneer Kit. https://www.cypress.com/documentation/development-kitsboards/cy8ckit-042-ble-bluetooth-low-energy-42-compliant-pioneer-kit
11. Raspberry Pi Foundation: Raspberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
12. Bluetooth Special Interest Group: Specification of the Bluetooth System, Covered Core Package Version: 5.0, Kirkland, WA, USA, December 2014
13. Amazon: Amazon Web Services (AWS) - Cloud Computing Services. https://aws.amazon.com/pt/
14. Amazon: 10-Minute Tutorials with Amazon Web Services (AWS). https://aws.amazon.com/getting-started/tutorials/
15. Oracle Corporation: MySQL: MySQL Workbench. https://www.mysql.com/products/workbench/
16. Postman, Inc.: POSTMAN | API Development Environment. https://www.getpostman.com/