Internet of Things

Group 3: Maxim-Leonid Rezan, Tiago Caldas De Silva, Elena Maria Pesci

Winter semester 2023-2024

Project: Thingy52 connected to Raspberry Pi through ESP32

The project aims to transfer data collected by the sensors of the Thingy52 through Bluetooth Low Energy to the ESP32. The information is then passed using the MQTT protocol to the Raspberry Pi and visualized in the ThingsBoard cloud service.

Hardware:

Thingy52

It is a compact, power-optimized, multi-sensor development kit device designed for collecting environmental data. Thingy can sense movement, orientation, temperature, humidity, air pressure, light, color, and air quality. The data can be transmitted via Bluetooth to Bluetooth-enabled devices.

J-Link EDU Mini

It is a minimalistic debug probe that has been used in the development of the project to flash code into the Thingy52.

ESP32

ESP32 is a versatile microcontroller and system-on-chip. It includes built-in support for Wi-Fi and supports Bluetooth Low Energy, which are both vital in our application.

Figure 1: Thingy52 [1]

Figure 2: J-Link [2]

Figure 3: ESP32 [3]

Raspberry Pi

The Raspberry Pi is a series of small, affordable, single-board computers (SBCs). It encapsulates all essential components of a computer in compactly.

Figure 4: Raspberry Pi 4 [4]

Development steps:

Note that whenever a Wi-Fi connection is mentioned in our report, we refer to the ‘max_a53’ hotspot from the personal phone of one of us. The choice is based on the ease of accessing IP numbers and useful information about connected devices, besides the possibility of changing location when working on the project.

⟶ Raspberry Pi setup:

We started working on the Raspberry Pi first. Since it is a computer, to function it needs a running operating system. Raspbian is an operating system tailored to work on Raspberry Pi, we proceed by uploading it on microSD using a laptop. The microSD was then inserted into the computer.

We don’t dispose of a keyboard or monitor to use with the Raspberry Pi. Therefore, we built a headless configuration for the Raspberry Pi. This procedure allows us to visualize the desktop of the computer directly on a laptop connected to the same Wi-Fi. This result was accomplished by the usage of TigerVNC and PuTTY.

The next step was to install the MQTT broker (mosquito) on the Raspberry Pi.

To be able to organize and manage the data from the communication, we installed ThingsBoard on the microcontroller. The platform runs on Java 11. The platform can be accessed by other devices connected to the same Wi-Fi, so we’ll show tables and graphs from our laptops.

⟶ Collecting data from Thingy52

The code uploaded to the Thingy52 has been written in VSCode, exploiting the functionality of the extension nRF connect and Zephyr.

The .h files contain the declaration of the functions, while the .c files implement them. Both are contained in the src folder of the ‘thingy52’ folder on GitHub.

The file ‘led’ includes the function turn_on_color used to light up the led, and the function for the led initialization.

The file ‘sensors’ collects information from the st_hts221, which can detect temperature and humidity. Three global variables are declared and will be used to store the values of temperature in Celsius

degrees, humidity percentage and timestamps, the latter is equal to the number of milliseconds since the system boots. These variables are used in the Bluetooth file.

The init_sensor function detects if the trigger is enabled, which means that the data is ready to be collected. If detection is positive it calls the function get_measuraments.

The function get_measuraments fetches the values of temperature and humidity and updates the respective variables, while also saving uptime in timestamp. Every time the get_measurament is executed the light of the LED turns green.

The file thingy_bt sets the Bluetooth configuration, defines and registers a service, and enables Bluetooth. Then, it verifies if it was successfully initialized in order to start the advertisements. Callback functions for connection and disconnection are also registered. Connection is emphasized by a yellow light and disconnection by a purple light.

The function notify_server sends data to the server without expecting an acknowledgment and is associated with the color red. It uses the variables created by the sensors (temperature, humidity, timestamp) through the service previously registered.

The main file calls the functions for the initialization of LEDs, sensors, and Bluetooth. It also contains a loop that notifies the server using notify_server every 3 seconds.

⟶ Communication between ESP32 and Thingy52

To load code on ESP32 we preferred Arduino IDE. The file to refer to is esp32.

The connection is created by exploiting the Bluetooth Low Energy technology. This approach is particularly effective since we are using battery-powered Thingy and the communication happens in short distances.

The setup function initializes the Bluetooth, sets advertisement callback functions, checks if the advertisement comes from one of our thingies, and if yes saves the reference of the device to connect later. Then, configures Bluetooth properties and starts the scan. When esp recognizes the specified number of devices (N_DEVICES) it stops the Bluetooth scan because all the desired devices were found.

The loop function contains a loop to connect to the preset number of devices, in case the connections aren’t established yet. The connection is executed using the function connect_ble_device. It creates a client and specifies callbacks to connect and disconnect (MyClientCallback class). It obtains a reference to the service and the characteristics of the connection. It uses the characteristics to see if the connection supports a notify, if it does it registers a callback to it, one for each device. The different notify callback functions allow us to distinguish the source of the data.

Each notify callback function specifies the device ID and calls the function notify_callback_func passing the data received, the length of the data, a boolean to specify if it’s a notification or not, and the device ID. Notify_callback_func verifies if the length of the data is sufficient to retain all the data (temperature, humidity, and timestamp) and publishes the data in JSON format.

The file also contains verification of connection with the PubSubclient and if needed reconnection, client loop, and delay.

⟶ Communication between ESP32 and Raspberry Pi

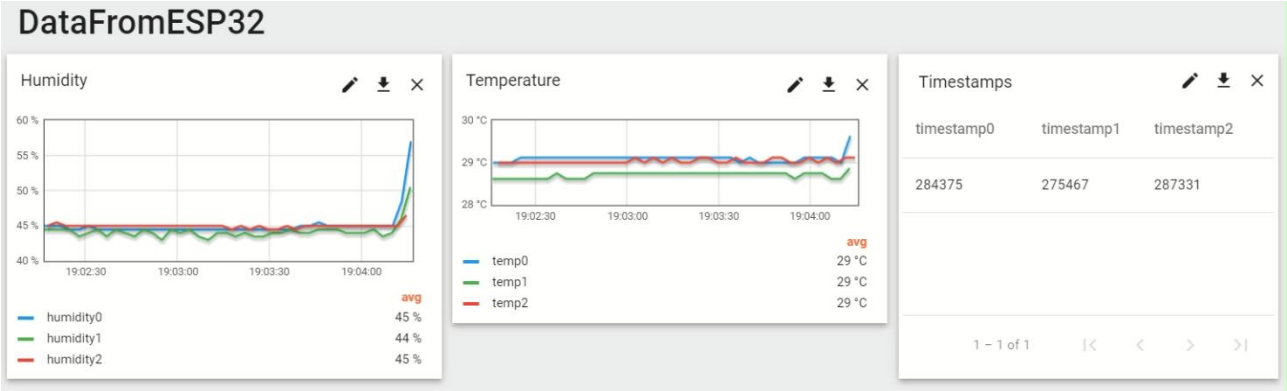
The aim is to pass the data received by Thingy52 to ThingsBoard using a MQTT transmission.

The MQTT protocol is based on a publish/subscriber messaging pattern. We created a topic through ThingsBoard named ‘telemetry’, which is accessible by the broker and ESP32.

The code loaded on ESP32 is written in ArduinoIDE using the libraries PubSubClient.h, stdio.h, and WiFi.h. The function setup contains the connection to the Wi-Fi, the connection to the MQTT broker, and the subscription to the topic. Inside the loop, we first ensure that the communication with the broker is abled, and then the data regarding the humidity and temperature is published on the topic. After the publication of the message, a confirmation of accomplishment is printed.

Data Visualization:

The platform ThingsBoard allows the creation of a dashboard where the information received by selected devices can be visualized in real time. We decided to use the widgets Entities Tables, which shows the latest value of the time stamps at which the detection occours, and Timeseries Line Chart, to plot the sequence of temperatures and humidities collected.



Final Conclusions:

It’s important to highlight a few of the main takeaways produced by the development of this project. The challenges have been many. For example, we encountered some difficulties in the installation of Raspian, and we spent a lot of time on the installation of a working version of Java 11. We divided the tasks to carry out among us, and when we combined the codes, they seemed too heavy for ESP32… it was just a wrong setting.

We learned to persist, exploit the useful information of forums and websites, and pursue alternatives until we reached the desired goal.

The requests are met, and the information collected from the sensors of Thingy52 flows through the ESP32 using a BLE connection and reaches the ThingsBoard platform using the MQTT protocol. It’s meaningful to specify that the number of Thingy52 devices that transmit data, and their MAC addresses must be known ex-ante, but the connection between Thingy and ESP32 can be interrupted and restored.

The video attached shows our result in action.

References:

Images sources:

[1] https://www.nordicsemi.com/Nordic-news/2017/06/nordic-thingy

[2] https://www.segger.com/products/debug-probes/j-link/models/j-link-edu-mini/

[3] https://www.digikey.at/en/products/detail/espressif-systems/ESP32-DEVKITC-VIE/12091811

[4] https://www.berrybase.at/raspberry-pi-4-computer-modell-b-4gb-ram

Code sources:

https://docs.zephyrproject.org/latest/hardware/peripherals/sensor.html#triggers

https://docs.zephyrproject.org/2.7.5/reference/kernel/timing/clocks.html?highlight=time#uptime

https://docs.zephyrproject.org/latest/hardware/peripherals/sensor.html#c.sensor_trigger

https://docs.zephyrproject.org/2.7.5/boards/arm/thingy52_nrf52832/doc/index.html

https://docs.zephyrproject.org/latest/samples/sensor/hts221/README.html