Elena Curti Matr: 185431 A.A. 2023/24

IoT Project – DoorBell

Introduction

This IoT project aims to create a doorbell. The main characteristic is the possibility of using two channels for the communication between the devices, to make the communication more reliable in case one channel "breaks" temporary for some reason.

In this report I will describe the used components, the behaviour of the project and the main choices I made during the implementation of it.

Components

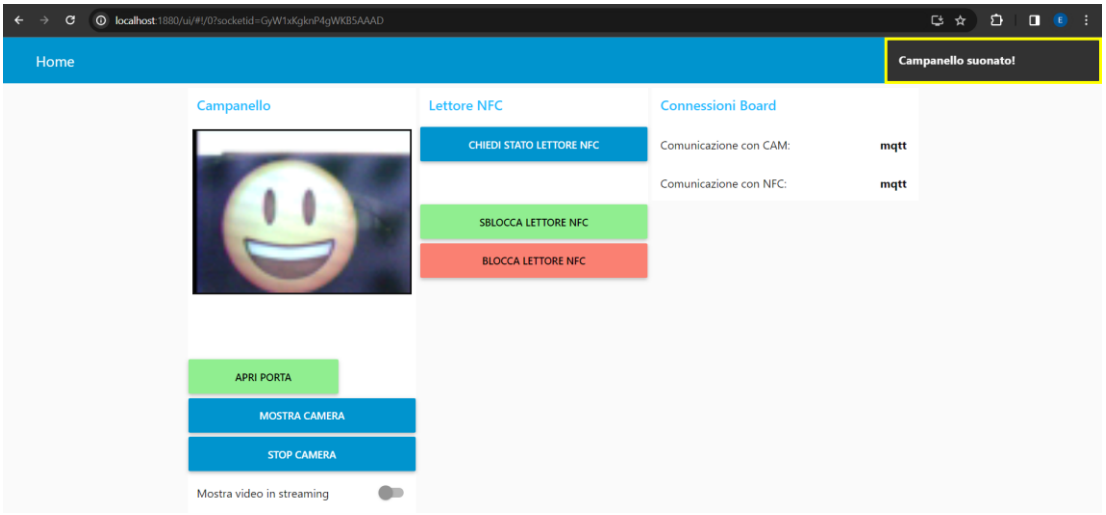
The hardware components I used are:

- The outside components (nearby the entrance door):
  - An ESP32 with:
    - An NFC reader (HW-147). This will allow the house owners to open the door.
    - A led to simulate the door opening
    - A button (the bell) that the guests can ring
  - An ESP32 with a OV7670 camera, to monitor who rang the bell.
- The inside components (inside the house):
  - A laptop, that is used as a display for the camera and as a central server

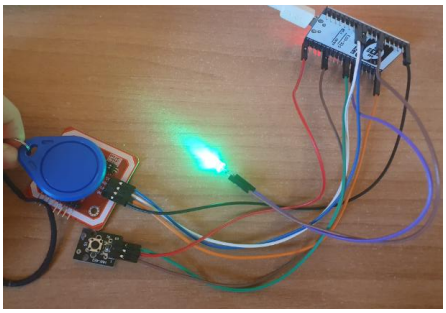
The outside components may also have been put in just one ESP. However, just sending one picture requires quite a big effort for the device, so I decided to use two separate boards.

Basic behaviour and main features

When someone arrives and rings the bell, the laptop inside the house gets a notification and shows the picture taken by the camera. From the laptop, the house owner can then decide whether to open the door or not, by pressing the corresponding button.



It's also possible to open the door from outside the house, using a NFC (authorized) tag:



The user can communicate some preferences to the devices such as:

- the authorized NFC tags
- the preference between seeing the live video or just one picture when someone rings the bell
- when the device should “sleep” (in order to save battery)
- when (and if) to enable the remote opening. In this case the communication happens with a remote server instead of a local one. The user can see the camera and open the door even if he/she is away from home (i.e. not connected to the same network as the devices)

Technology used

Wi-Fi and MQTT

The communication between all the devices happens mainly with Wi-Fi and MQTT.

I chose Wi-Fi because it has high data rate (useful when the user wants to see live video from the camera) and all the components already have it integrated. Also, since this is a project destined to domestic use, it’s easy to integrate it in a house if there already is a Wi-Fi network.

Then, as messaging protocol, I chose MQTT because it’s lightweight, it has different quality of services (useful to distinguish between reliable and unreliable communication) and features like *will message* and *clean session* that I used to determinate the state of a device in the network. Also, there is the possibility to use remote broker so that the user can communicate with the boards even if all the devices are not in the same local network.

Regarding MQTT, the following tools have been used:

1. Mosquitto. I used it to have a local broker on the laptop. Besides the publish/subscribes features, it provides a “clientid_prefixes” feature to allow connections only from clients whose id start with a specific string. This will be used to simulate a break between one ESP and the broker.
2. PubSubClient. I used this library in the ESPs for publishing and subscribing to the MQTT topics. It also provides a way to publish large messages (used for the photo). It doesn’t provide a way to publish messages with QoS different from zero. However, this is not a problem, because the ESPs never publish messages with QoS of level 2, but only receive them. Also, this library allows to specify the *clean session* and *will message* values.

Bluetooth

In this project I wanted to provide an alternative way to make the devices communicate, so I chose Bluetooth. Since it has low data rate, I chose to keep Bluetooth as a secondary channel for communication, and not as the primary. Like the Wi-Fi, the Bluetooth has short range, so this project can only be used if the devices are not too far away one another. This is a bit limiting but this project can be considered as a “model”, so it can be re-arranged according to the necessity (for example using LoRa if the devices are too far away).

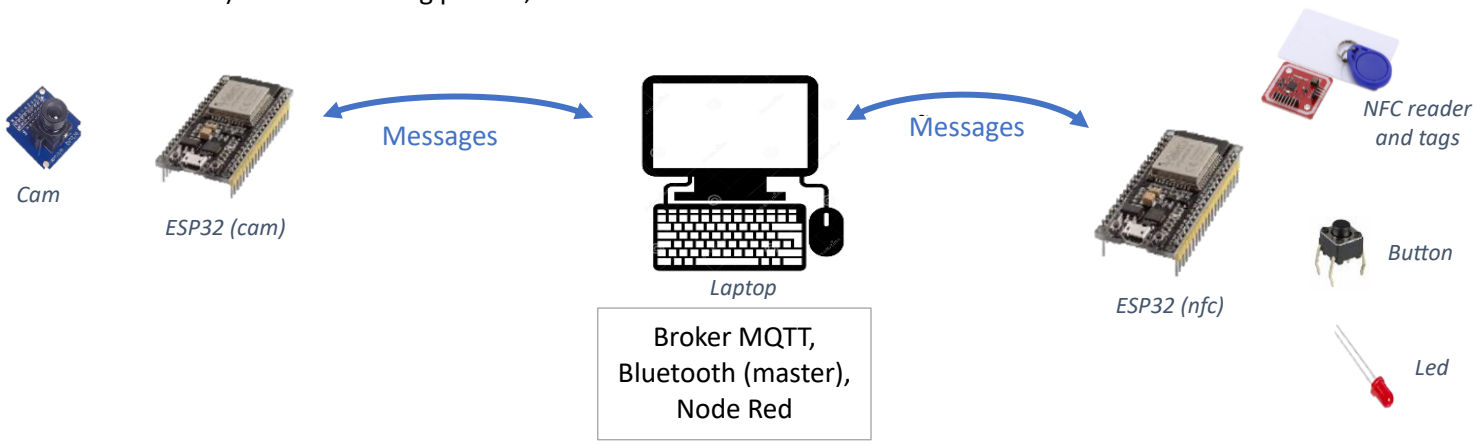
To integrate Bluetooth in the project, I used a python script that uses bluetooth module (for the pc) and BluetoothSerial library for the ESPs. These libraries provide an easy way to receive and send data. The python script also provides a way to connect to the desired devices and it's easy to integrate with Node-Red.

Node Red

In the project I wanted to provide an application that allows to see the camera images, open the door, enable/disable the NFC reader. I decided to use Node-Red since it provides a web interface, and it can be easily integrated in a web server. In Node Red it's also easy to integrate MQTT communication and the execution of python code.

Communication architecture

The communication of the devices happens as shown in the following picture. The two ESP devices communicate with the PC (where there is the MQTT broker, Node Red, and the Bluetooth script that connects to the devices). The ESPs never communicates with each other directly (making this a star architecture). In the following picture, the devices are all in the same Wi-Fi network.



If the user decides to enable the "remote opening", MQTT broker and Node Red aren't executed in the PC anymore, but in a remote server. In this case it's not mandatory anymore for all the devices to be in the same Wi-Fi network and the user can access the Node Red application also from another device that is not the pc (for examples a smartphone).

Configurations

In the following, I will refer to "configuration" meaning all the possible customization that the user can communicate to the devices. The configurations can be sent for example using mosquitto_pub (in this case they will only be communicate if the devices are communicating with MQTT) or Node Red (in this case they will be communicate even if the devices are communicating with Bluetooth).

The user can send some configurations to the ESPs. I distinguished between "global" and "local" configuration: the global are sent to all the devices, while the local just to NFC or the camera device.

The global configurations allow to change:

1. the MQTT broker. In this way, the user can communicate to the devices to disconnect from the local broker and connect to a remote one (or vice versa). The default broker is the laptop's IP address in port 1883.
2. the number of attempts to connect to Wi-Fi and MQTT broker before resetting. The boards are configured to try to connect to Wi-Fi and MQTT broker. If they fail for more than 20 times, they will reset. This number can be changed with this global configuration. To deactivated it a -1 must be sent (in this case the devices won't reset).

These two configurations can also be combined to always communicate with Bluetooth. If the user sends as global configurations (1.) an unreachable ip and (2.) the number -1, the two boards will power on the Bluetooth and communicate with it. However, communicating with only Bluetooth is not recommended, especially for the camera because sending the pictures is quite heavy.

Regarding the "local" configuration, for the camera, the user can decide:

- The frequency to send images. The user can specify to show the live video, instead of a single picture when a guest rings the bell. This frequency represents the sending distance between one image and the next one (in millisecond). To reset this and have just one picture, the user can send the number -1.
- Timeout for the video. If the user has decided to show the live video, he/she can decide to stop it after a certain number of seconds (default is 30). To have a video going on for ever, the user can send -1.
- The user can decide when to power on the Bluetooth. The possibilities are: never (a *0* must be sent); after three failed attempts of connection to MQTT or Wi-Fi (default behaviour, or sending *1*); always (sending *2*). This configuration only regards the powering on/off of the Bluetooth in the ESP. For example, the last option (always) is recommended if the MQTT link is known to be easily broken. In this case the Bluetooth is always on and the board doesn't wait the three failed attempts. If the ESP is connected to the MQTT broker and with another Bluetooth device, the devices will still communicate with MQTT, since it's the preferred way.

The configurations are sent in topic *my_devices/esp_cam/config.* Example:

```
{
    "freq_send_img": -1,
    "timeout_send_img": 30,
    "use_bluetooth": 1
}
```

Regarding the NFC reader, normally, if an NFC tag is read and it's in the list of authorized tags, then the door is opened. If a wrong tag has been swiped for 3 times, then the NFC reader stops reading for 30 seconds. After 5 wrong attempts, the reader is disabled (it stops reading tags), and only the user can re-enable it. Every time a tag is swiped, a message with the tag is sent. The configurations in the device with NFC are:

- Authorized tags (in a JSON array)
- Numbers of wrong attempts before disabling the reader (the numbers 3 and 5 as explained above)
- A true or false flag to specify if the comparison between the swiped tag and the authorized tags must be done in the device (default is true). Every time a tag is swiped, a message with the tag is sent, so this flag allows to keep the logic outside the device. For example, let's assume the user wants to disable the reader for 10 seconds after 2 wrong attempts and then for 30 seconds after every other wrong attempt. This can't be communicated to device. So, the user can decide to set this flag to false and keep the (custom) logic outside of the device. For example, it can be performed in Node Red. The user must then write his/her custom algorithm that possibly sends another message to the device to communicate to open the door.
- Seconds of inactivity after which the board will deep sleep. If no one rings the bell or swipe the tag for a certain number of seconds, then the board will sleep. When the bell is rung, the device will wake up. To deactivate this a -1 must be sent.
- The powering on/off of the Bluetooth (same as the camera)

The configurations are sent in topic *my_devices/esp_nfc/config.* Example:

```
{
    "tag_autorizzati": [
        "122.48.29.217",
        "209.53.34.217"
    ],
    "num_tentativi_errati": [3, 5],
    "check_tag_localmente": true,
    "use_bluetooth": 2,
    "secondi_board_inattiva": 60
}
```

The MQTT topics hierarchy

The first root level is *my_devices*. I used this level mainly to have a point of access of the devices in the remote broker.

The second levels are:

- *global_config* (where the globals configurations are sent; they are received by both the devices)
- *esp_nfc* (where the device with the NFC publishes and receives messages)
- *esp_cam* (same but for the device with the camera)

Let's now explain the third levels.

The NFC device can receive messages in the following topics:

- *config*. In this topic the device receives the configurations in JSON format.
- *reset*. If the payload is 1, then the device resets.
- *deep_sleep*. The payload must be in the format "number unit". It represents how much time the board can (deep) sleep. The unit can be s (for seconds), m for minutes and h for hour. By sending for example "1 h" the user communicates to the device to sleep for one hour.
- *led.* If the payload is 1, then the device opens the door.
- *nfc_reader_state*. If the payload is 0 then the NFC reader is disabled. If it's 1 then it's enabled. A 2 in the payload is intended as "tell me the state of the NFC reader", so the device will publish a message in /my_devices/esp_nfc/*nfc_attempts* topic in the format "*state n*", where state is 0 if the NFC reader is enabled and 1 otherwise. The n is the number of wrong swiped tags performed until now.
  For example, if the user wants to know if the NFC reader is currently enabled or disabled, he/she must send a message in /my_devices/esp_nfc/*nfc_reader_state* topic with payload 2. The device will then respond, for example, "1 5" meaning the reader is disabled (1) because 5 wrong tags have been swiped.

The NFC device will publish messages in the following topics:

- *state.* This is a flag that represent wheatear the board is connected to the MQTT broker or not. It will be explained in detail in the next chapter.
- *button*. If the bell is rung, a 1 will be published in this topic.
- *tag_swiped*. When a NFC tag is swiped, a message with the tag content is published here.
- *intruder*. When too many unauthorized tags are swiped, a message with the tag content will be published. This is sent right before disabling the reader.
- *nfc_attempts*. Described previously.

The camera also receives messages regarding configurations, reset and deep sleep (but in the *esp_cam* second layer topic). It will also publish its state in the *state* topic. The camera device is subscribed to *my_devices/esp_nfc/button*, *my_devices/esp_nfc/intruder*, *my_devices/esp_cam/request_send_img* and will publish a photo whenever it receives a message in these three topics. The photo is published in *my_devices/esp_cam/image* topic.

MQTT details

For the "normal" communication (image bytes, button pressed, tag swiped, …) I used level zero and unretained messages.

For the configurations, that are more important, I used level two QoS and I enabled the retain flag. In this way I'm sure the configurations are received even if the device will reset or disconnect to the broker.

To check the state of the connection between the two ESP and the MQTT broker, I used *Will messages* with *retain* flag. As soon as the device with the camera connects to the broker, a retained message with payload "1" is published in topic *my_devices/esp_cam/state*. In the same topic I set a *Will retained message* with payload "0". Now, to check the state, it's enough to subscribe to the previous topic and check If the retained message is one or zero. When (and if) the device with the camera disconnects ungracefully, then after a few seconds the *Will message* will be sent to the client (Node Red) subscribed to the topic. This mechanism is used in Node Red to switch to Bluetooth and to communicate to the user which devices are connected with a MQTT connection.

The same was done with the device with the NFC, but in the topic *my_devices/esp_nfc/state.*

Switching between MQTT and Bluetooth

The switch between the two channels of communications is performed by Node Red. During execution, Node Red periodically checks if the connection with the MQTT broker is established:

- In case it is, then the application will eventually receive the *state* retained messages of the camera and the NFC.
    - If the message is "1" then it will possibly stop the Bluetooth connection and start communicating with MQTT.
    - If it's "0", the device isn't connected to the MQTT broker yet, so Node Red will start (or continue) the communication with Bluetooth.
- Otherwise, it will start the python script to connect to both of the devices using the Bluetooth. The ESPs tried the MQTT connection and probably failed too, so after three attempts the ESPs power on the Bluetooth[1]. After a few seconds, Node Red will connect to the devices and all of them will communicate with Bluetooth from now on.

The check of the connection with the MQTT broker is performed periodically, even if the devices are connected with Bluetooth. This is done both by Node Red and the ESPs. This aims to re-establish the MQTT connection as soon as possible, since MQTT is the preferred way to communicate and the break of the link may be temporary.

It may happen that one ESP is connected with MQTT to the laptop and the other one with Bluetooth. In case one of the ESP sends a message with Bluetooth, Node Red will possibly forward it also to the other ESP, using MQTT (and vice versa). This only happens if the message sent by the first ESP must be received by the other ESP too. In the Node Red dashboard, the user can see that the two devices are communicating with different technology, but the application and the communication are still working.

[1] This is the default (and recommended) behaviour, but the powering on/off of the Bluetooth can be changed like described in the configurations.

Of course, if for some reason both MQTT and Bluetooth don't work, then the devices cannot communicate anymore. In this case, the devices will try to "self-fix" themself by performing reset (after a few attempts).

Communication performance

Time

Regarding the communication performance, the "small messages" (i.e. the non-picture messages, that generally have less than 100 bytes) are usually sent in a few seconds by all devices. Instead, I noticed that the publishing of "big messages" (i.e. the pictures, that are around 19 Kbytes in size) with Wi-Fi require at least 100 milliseconds and up to 30/35 seconds. Also, if the network isn't stable enough the publishing with MQTT will probably fail. I think this is due to a limitation of the device itself, since it doesn't occur when the ESP sends small messages nor when a more powerful device (the laptop) sends big messages. I also tried using a different library (MQTT client) but it didn't fix the issue.

The difference in time between using Wi-Fi or Bluetooth is generally negligible. Regarding big messages, the ESP often can't send more than 5/6 pictures in one time using Bluetooth. So, as expected, the live video should be disabled if the devices are using Bluetooth.

Space

Regarding the distance of the devices, I noticed that for the Bluetooth it's necessary a maximum distance of 8 meters. For the Wi-Fi it's a little more (around 15-20 meters).

Future work

To conclude, to make this project a "real-life" doorbell, some further improvements must be done. These improvements mainly concern the resolution of security problems, such as authentication and cryptography of the messages. These problems weren't discussed in this project because they weren't the goal of it.