# Internet of Things - Final Project Report

Authors: Alessandrini Luca (10569363), Renzi Flavio (10659808)

## Contents

## 1 Approach

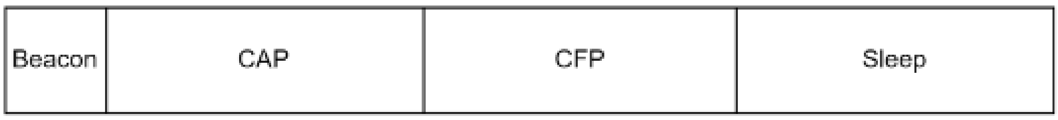


Figure 1: Structure of the beacon interval.

The protocol we created implements three types of messages:

- `BEACON`, containing all the information about the structure of the beacon intervals,
- `REGISTER`, used to allow a client to register to the PANC,
- `DATA`, used by sensor clients to send data to the actuators ones.

### 1.1 Beacon Message

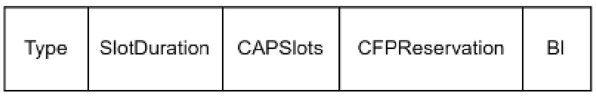


Figure 2: Structure of the Beacon message.

Contains all the information for the correct alignment and reservation of the slots. it is composed of:

- `Type`: `BEACON`,
- `SlotsDuration`: the dimension of a slot in seconds,
- `CAPSlots`: number of slots in the CAP part,
- `CFPReservation`: an ordered list of addresses that represent the allocation of slots in the collision-free part,
- `BI`: the duration of the entire beacon interval in seconds.

### 1.2 Register Message

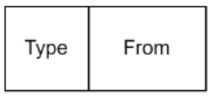


Figure 3: Structure of the Register message.

This message is sent by new clients in the collision access part to join the network and obtain a reserved slot in the collision-free part. It only consists of two fields:

- `Type`: `REGISTER`,
- `From`: the number corresponding to the client id.

### 1.3 Data Message

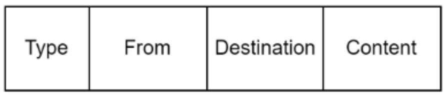


Figure 4: Structure of the Data message.

This type of message is the one sent by a sensor client. It is composed of:

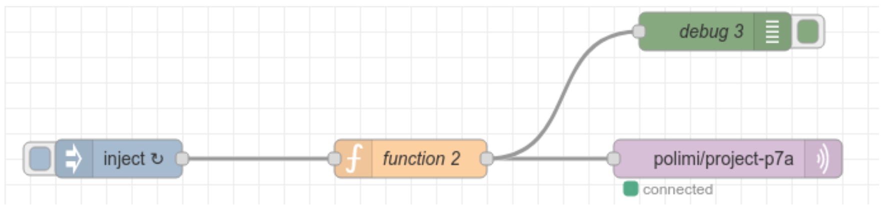
- `Type`: `DATA`,
- `From`: address of the sender,
- `Destination`: address of the client we are sending to,
- `Content`: data contained by the message (in our case the humidity percentage).

## 2 Components

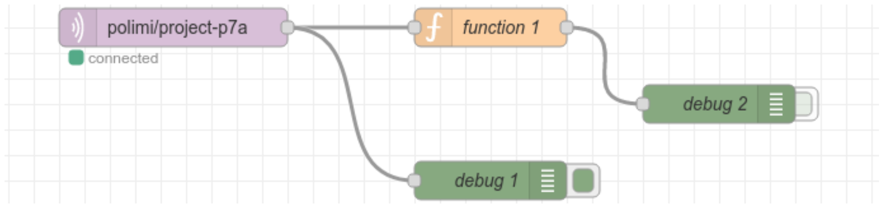
### 2.1 PAN Coordinator

The pan coordinator is developed using node-red.

The sending of the beacon is handled by the flow in figure 5a, and is triggered by the inject node which configures the parameter of the beacon. After that, a function node is used to access a global queue to schedule the slot reservation so that when we have to send the beacon we can access the queue to send the reservation. At the beginning, we have only the the beacon slot and the CAP part that allows clients to register. All the other slots are assigned to the sleep part, and while clients send `REGISTER` requests, slots are moved from Sleep to CFP. Once the sleep part has no more available slots, clients are scheduled in a



(a) *Flow responsible of the structure of the BI.*



(b) *Flow responsible of the handling of* `REGISTER` *requests.*

Figure 5: Node-red flows of the PANC.

round robin way. To better understand this, let's make an example: suppose we have 2 slots in the CFP, and the sleep is composed of zero slots. Suppose five clients are registered: A,B,C,D,E. The schedule for the CFP part in the first beacon will be: A,B. The one for the next beacon will be C,D; then, E,A; and so on.

In another flow (the one in figure 5b), we subscribe to the same topic to read if a device sends a `REGISTER` message in the collision access part and add them to the global queue to be scheduled.

## 2.2 RFD Devices

The client has an internal flag to store whether has been already registered.

When it boots, it subscribes to the same topic of the coordinator waiting for a beacon. One received a beacon, the client knows when the CAP is, and, waits its beginning before sending a `REGISTER` request at a random slot in the CAP part to minimize collision probability. Notice that we don't wait the end of the beacon slot, since we are assuming that the client is able to decode it only when the message has been fully sent (i.e., at the end of the beacon slot). After that, the client will be considered for the next beacon interval schedule.

### 2.2.1 Actuator Device

The ID of this device is either 3 or 4.

After the registration phase, it listen for messages, and when it receives a percentage of humidity directed to him, it sets that percentage as the luminosity level of the LED connected to it.

The representation of this sensor on WOKWI is visible in figure 6b.

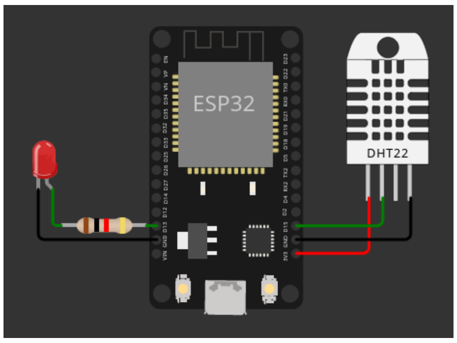
### 2.2.2 Sensor Device

The ID of this device is either 1 or 2.

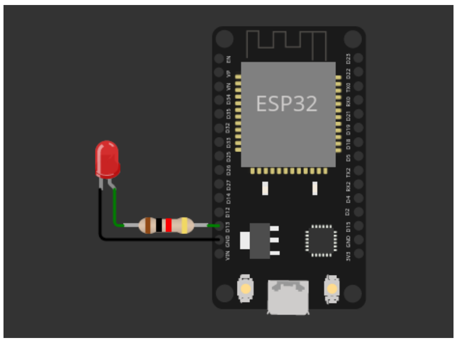
After the registration phase, at every beacon, it checks whether it is scheduled in the CFP. If it is, then it computes the waiting time before sending the humidity sample which is simply computed by multiplying the number of slots to wait by the duration of a single slot. The number of slots to wait is computed by performing the sum of:

- Number of slots in the CAP,
- Index of the client in the scheduling array.

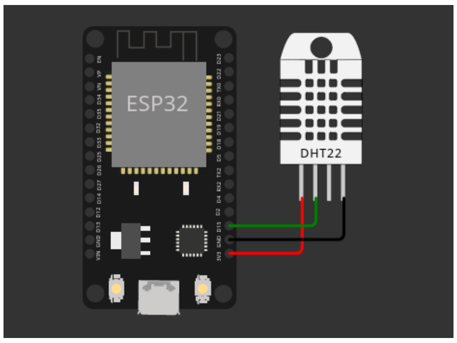
We don't consider the slot of the beacon in the waiting time, since we are assuming that when the client is actually able to decode the beacon message, the slot is actually ended.



(a) *General Function Device*



(b) *Actuator Device*



(c) *Sensor Device*

Figure 6: Implemented clients.

The representation of this sensor on WOKWI is visible in figure 6c.

#### 2.2.3 General Function Device

We faced the problem of WOKWI running slower than actual time. In order to do lighter tests, we also created two of these devices (with ids 5 and 6). These devices are capable of both setting a led and sending humidity values. These clients can moreover be configured to work both as an actuator device or a sensor device by setting the variable `CLIENT_TYPE`. Additionally, the client can have both parts (actuator and sensor) working together. This is very helpful in the testing phase since it combines the way of reasoning of the two device types analyzed above.

It can be used as a simple sensor client by setting `CLIENT_TYPE` to `HUMIDITY` or as an actuator by setting it to `LED`. If set to `ALL`, it behaves simultaneously as a sensor and an actuator client. If 5 needs to send a message, it will put as destination 6, and vice versa.

The representation of this sensor on WOKWI is visible in figure 6a.

## 3 WOKWI's simulation throttling problem

### 3.1 Note on time checks

We cannot check that messages arrives actually in the CAP slots, or that they are actually sent in the reserved CFP slot (or CAP), for two main reasons:

- We are relying on MQTT so the broker could have some little delays,
- WOKWI doesn't run at 100% of speed (we are at about 25% on linux, and 50% on windows). For this reason, we also cannot actually sleep, because the client scheduled in the last CFP slot could send the message at a moment in time that the destination client could consider "sleep", just because maybe was running a little bit faster.

Anyway, if we could run clients on real devices, and if we assume that MQTT has no delays due to the broker, clients are implemented in a way that guarantees the send of a register message in the CAP slots, and to respect their CFP schedule.

## 3.2 Notes on speeds and rates

Since we had problems due to WOKWI speeds (as said in section 3.1), we decided to fix in advance the duration of the Beacon Interval and of a single slot. These times have been set in order to allow messages to be sent and received correctly even if clients were throttling (hence, not running at real speeds).

Due to this, we decided to obtain $r$ and $R$ starting from the parameters needed to run the simulation ($T_s$, $BI$, number of slots in CAP and CFP, ...).

We decided to set the dimension of the beacon as size of the message that fits one slot (i.e., as assumption we considered that the beacon fits exactly the slot, and we can pad smaller messages to reach that size).

Considering the sizes of its sections (represented in figure 2):

- `Type`: 2 bits,
- `SlotDuration`: 32 bits,
- `CAPSlots`: 6 bits,
- `CFPReservation`: $8 \cdot 57 = 456$ bits. 8 is the length of the address, and 57 is the maximum number of slots in the CFP part when considering CAP of dimension 2,
- `BI`: 32 bits.

This leads to a size of the beacon message equal to:

$$2 + 32 + 6 + 456 + 32 = 528 \text{ bits} = 66 \text{ Bytes}$$

Setting $T_s = 0.5$ (due to limitations of WOKWI's simulation), we obtained a nominal rate of:

$$R = \frac{528}{0.5} = 1056 \left[\frac{b}{s}\right]$$

Considering the dimension of the `DATA` message (figure 4):

- `Type`: 2 bits,
- `From`: 8 bits,
- `Destination`: 8 bits,
- `Content`: $528 - 8 - 8 - 2 = 510$ bits. This guarantees that the messages have the same size.

We decided to include a single humidity reading for each message and use padding for the remaining bits. In a real scenario, we may want to do more readings within the $BI$ interval (if the duration is really long), in order to use them maybe for other purposes (for example to perform an average before setting the LED). This would make useful bits that now are dedicated to padding.

Thus, we can compute $r$ by supposing $BI = 30$ seconds, as:

$$r_{\min} = \frac{L}{BI} = \frac{528}{30} = 17.6 \left[\frac{b}{s}\right]$$

The register message is the smallest one: it has only `Type` and `From` fields: for this reason it will be padded similarly to fit one slot.