**Sending data over WiFi: Connected Bird Nest**

In the high-power mode, all the data is sent through WiFi, Sigfox is not used. Furthermore, pictures need to be sent along with the sensor data. Here is a brief explanation about how the system works.

Our whole system is driven by our main MCU (Microcontroller Unit), the STM32L0, which we call L-zero for short. To connect our system to the WiFi, we have a WiFi module called the ESP32-Wrover, which we call the Wrover for short. The Wrover module contains itself an esp8266 microcontroller that we can reprogram, so it can have its own decision-making processes. This is especially useful in the case of the picture taking, because it is a process that consumes a huge amount of power and transmits a huge amount of data, so if the Wrover can take the decision on its own to take pictures without going through the L0, we can save a significant amount of power.

The L0 is interfaced with all the other sensors and gathers and stores their data, as well as formats the data using the encoding detailed in the Sigfox-encoding document. When it is time to send a message, the L0 will send the encoded data to the Wrover, which is expected to immediately send that data to our backend through the WiFi.

To summarize, we have two kinds of messages when using WiFi:

- Pictures, sent on the Wrover’s decision when it is deemed useful to take one

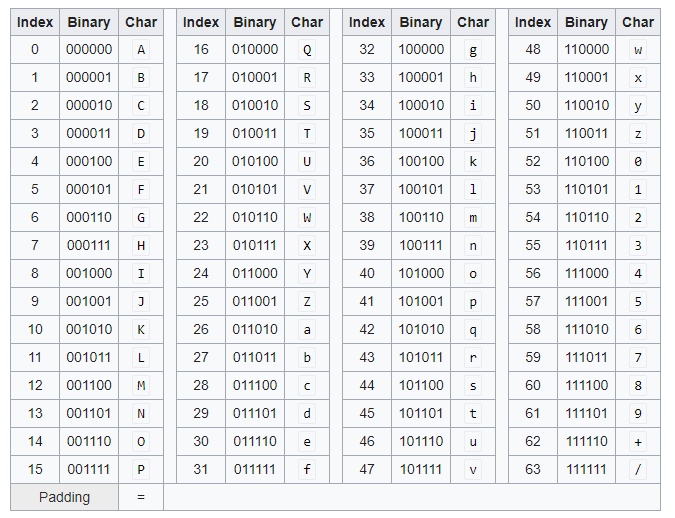
- Sensor data, sent by the Wrover on the L0’s decision, possibly once a day (but maybe we can afford a higher frequency in this high-power mode, this will need to be tested and is not crucial right now)

Both of those messages, we send using HTTP requests.

Pictures

When the Wrover takes the decision to take a picture, it immediately encodes it into a base 64 string, this is a way to encode a picture into text without taking too much space. Each character corresponds to 6 bits (unlike hexadecimal where each character corresponds to 4 bits).

Here is the conversion table that is used:



There are a few things to be aware about here, most of the characters are alphanumeric, but the last two are “+” and “/”. When we will transmit the picture data through an HTTP request, it will be in URL-encoding, and the characters “+” and “/” are not friendly with URL-encoding, thus, for the transmission, they are replaced by “%2B” for “+” and “%2F” for “/”. Upon reception and decoding of a picture message, those characters need to be changed back to “+” and ”/” before attempting to convert the base64 to an image, otherwise the conversion will not work.

To send the picture to our backend, we use a POST request with header “Content-Type: application/x-www-form-urlencoded” and the URL-friendly base 64 string in the payload.

Sensor data

To encode the sensor data, we use the same encoding as for Sigfox messages, because it is the most compact way to transmit the data of all the sensors, this can save a small amount of power at each transmission, and it is simple because we can use the same encoder and decoder for WiFi and Sigfox.

We use the same kind of POST request to send that data to the backend without changing it.

Notes about visits:

In case we use the approach of dividing the day into periods, that makes us use two Sigfox messages, but the HTTP request doesn’t have the 12 bytes restriction, allowing all that data to be sent in one request. We can then get rid of the identification bits and concatenate both Sigfox messages into one HTTP request. The backend will then have to parse that data accordingly upon reception.

Notes about message recognition:

Since we have two types of HTTP requests, the backend has to be able to recognize which is which. This can easily be done by looking at the length of the payload. If it is absurdly long, it’s a picture, if not, it’s sensor data. We can formalize this idea in code later.