GENTILLON Loris
VEGREVILLE Thibaud
ZHENG Jian

RICM4

REPORT
Social Network Lora (with ESP32-LoRA pods)

Year : 2017 - 2018
Teacher : RICHARD Olivier

DIGEST

Project Presentation

This project aimed to develop a framework for self sufficient system that would allow the user to communicate through a low frequency LoRa network. The user would connect to the pod through a mobile application. The pod has then to process the information to others pods on the network, in order of having a stable state.

The idea is to have an independant (in the sense of non-existing infrastructure reliant) system that would be use in particular environments, such as cellular-less area (desert, mountain).

We have chosen to separate our efforts with the following :

- ➢ Develop a mobile application that would connect to an LoRa pod thanks to wifi. This application had to send/receive the data to the pod, this is the front end part of our system. We have chosen a survey application to demonstrate the system.
- ➢ Develop a micro-server hosted on a LoRa pod hardware, able to create a wifi LAN and proceed LoRa data. The pod is the key of the idea, by creating the bridge between two (or more) distant point running the same application.
- ➢ Develop a LoRa protocol for this application, including a standard for communication.

Software and tools

LoRa

LoRaWAN (Low Power Wide Area Network) is a radio technology capable of sending a small amount of data (nearly 30kb/s) on wide range (around 15km at last). The main interest of this technology is the capability of send data on a quite decent range in area non cover by internet network or telephonic operator.

Angular / Ionic

For the application, the choice of developing with angular and ionic has been done because of the compatibility with both IOS and Android world. Because this application doesn't aim to use lower level or device specific function, it simply suits our needs.

Angular takes care of all the backend functionality of the application, such as which text to show on screen, dynamic generation of option, and communicate with the pod.

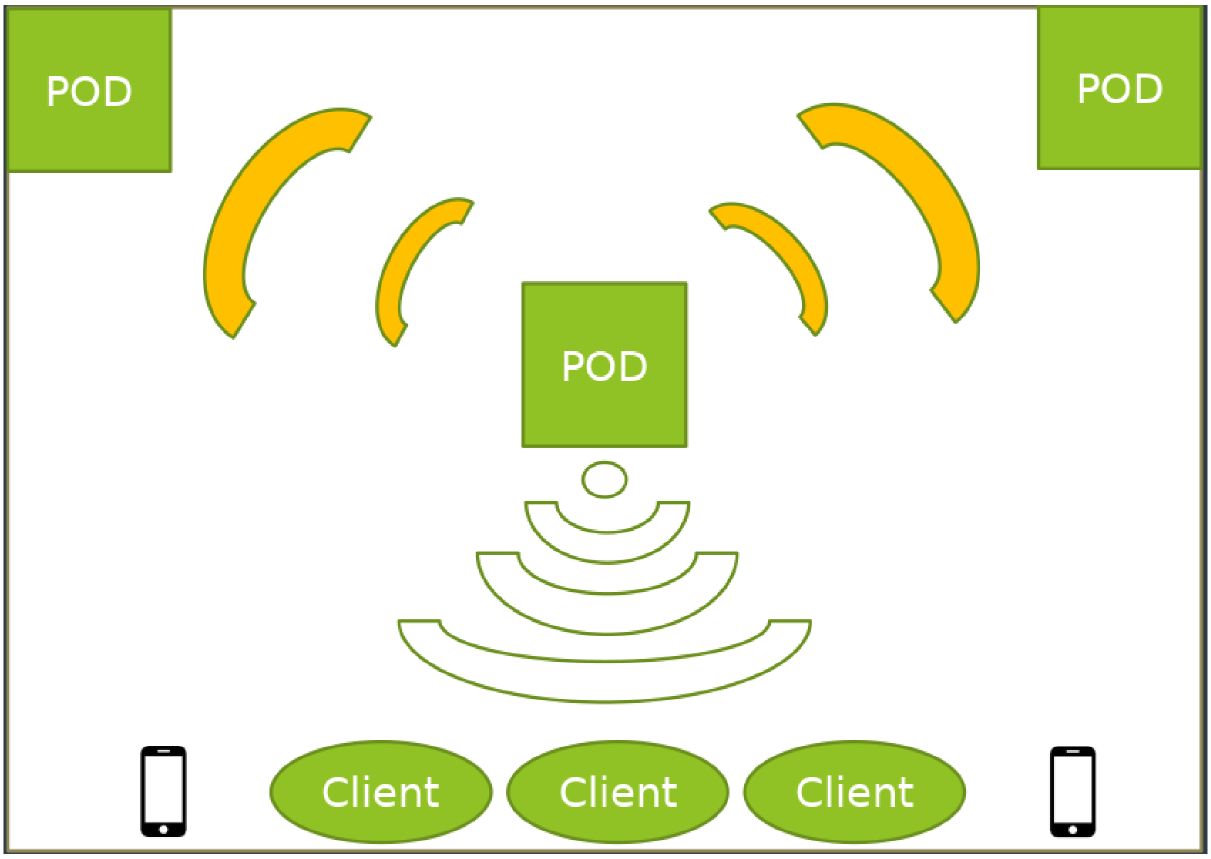
Ionic handle the frond end, with graphical component, animation, and ordering the pages accordingly.

Json

The data the pod and the application exchange are following a json format, that we wanted to be as generic and as vast as possible, in order of reusability.

Operation

Overall



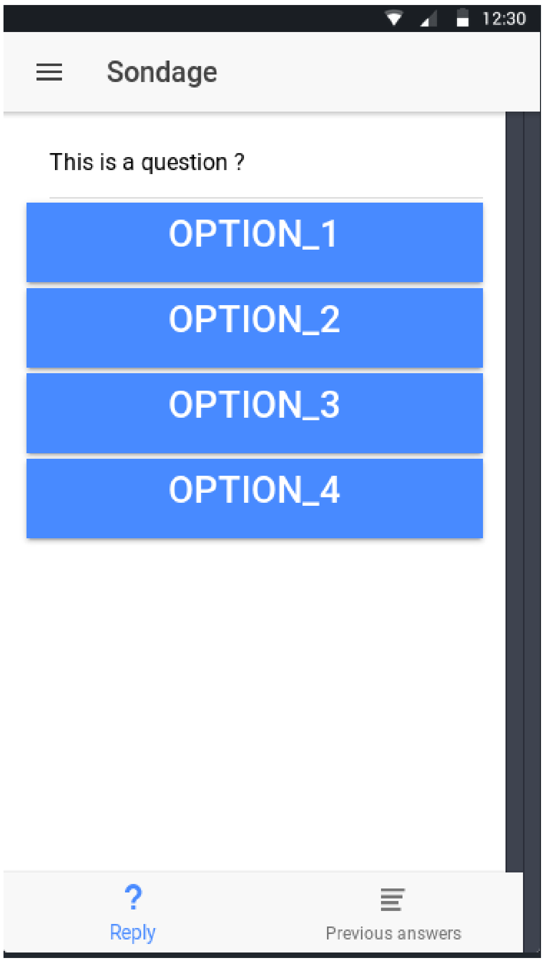
The main goal of this global algorithm is to run different kind of application on the same model. All of this application share at least the same principale : have a local network from which you can't connect to internet globally, and still can communicate with other local network that are running the same application. It could be, for example, survey application, security network, humanitarians camps, etc...

The functioning is the following :

- An application is running locally on the network, collecting data from all users (votes in the survey example).
- A trigger is reach (depending of the application) at a given time (in the survey example, it's an amount of votes).
- The LoRa pod broadcast its information all around, to the other pod running the same application.
- All this pod update themself with incoming data from other pod, and spread these modification in a flooding-like mechanism.

Sending / receiving data

Application-side



The application sends / receives data with http GET request.
For instance, on the initial load of the application, there is a GET request to the pod, to get the question and the options in order of building them dynamically.

To receive data with angular, the easiest way is to declare a variable that will contains the previously described json, and then to use the get request, in order of getting an observable and then subscribe to this observable to feed the variable.

Thanks to angular 4 , there is the possibility to cast (with an interface) the observable instead of maping it manually. Which result in this :

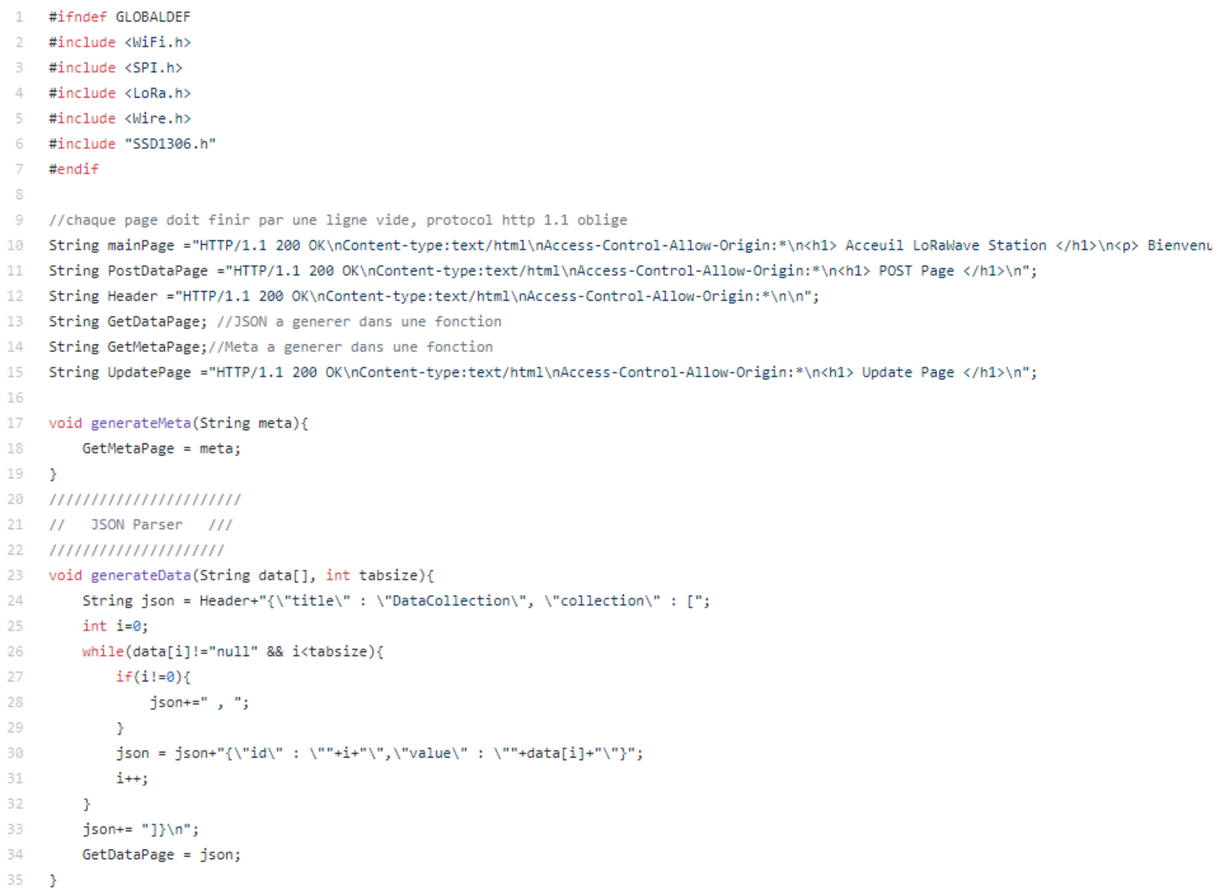
```
getResult() : Observable<Question_Result>{
  return this.httpC.get<Question_Result>(this.urlGet);
}
```

```
setResult(){
  this.getResult().subscribe(res => this.retResult = res);
}
```

which is quite small yet fully functional.

LoRa pod-side

From LoRa pod side, it’s a quite more challenging code that handle the data feature. In fact, the pod is divided in two section : The wifi one, and the radio one.
The wifi side handle an hand-craft local web server, that can serve on request some page and, using this way, JSON data. Consequently, we have found this way the more efficient to communicate with the application. On GET request, we send JSON data resuming the pod situation, and we update information of the pod on POST request. At least, the functioning is quite similar to an online web API.



```
#ifndef GLOBALDEF
#include <WiFi.h>
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306.h"
#endif

//chaque page doit finir par une ligne vide, protocol http 1.1 oblige
String mainPage ="HTTP/1.1 200 OK\nContent-type:text/html\nAccess-Control-Allow-Origin:*\n<h1> Acceuil LoRaWave Station </h1>\n<p> Bienvenu
String PostDataPage ="HTTP/1.1 200 OK\nContent-type:text/html\nAccess-Control-Allow-Origin:*\n<h1> POST Page </h1>\n";
String Header ="HTTP/1.1 200 OK\nContent-type:text/html\nAccess-Control-Allow-Origin:*\n\n";
String GetDataPage; //JSON a generer dans une fonction
String GetMetaPage;//Meta a generer dans une fonction
String UpdatePage ="HTTP/1.1 200 OK\nContent-type:text/html\nAccess-Control-Allow-Origin:*\n<h1> Update Page </h1>\n";

void generateMeta(String meta){
    GetMetaPage = meta;
}
//////////////////////
//   JSON Parser   ///
/////////////////////
void generateData(String data[], int tabsize){
    String json = Header+"{\"title\" : \"DataCollection\", \"collection\" : [";
    int i=0;
    while(data[i]!="null" && i<tabsize){
        if(i!=0){
            json+=" , ";
        }
        json = json+"{\"id\" : \""+i+"\",\"value\" : \""+data[i]+"\"}";
        i++;
    }
    json+= "]}\n";
    GetDataPage = json;
}
```

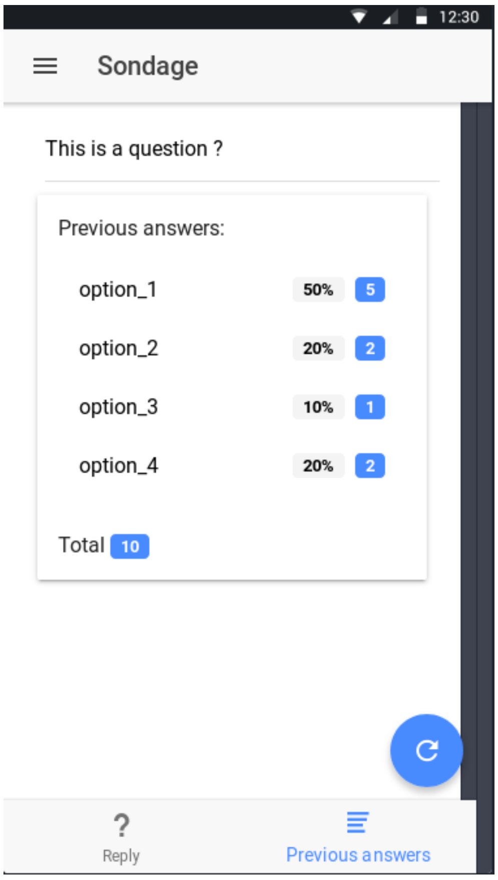
Exemple of the web server

On radio side, it’s more easier than it seems : In fact, a local callback handle all the stuff related to the message catch. At least, we only have to handle the message treatment.

Processing data

Application-side

For this prototype, the data processing is quite small. Everything is done by knowing whether the variable retResult is set or not, and by displaying accordingly informations. The only real processing, beside visual, is calculating the percentages based on the previous votes, which ain't heavy.



This is a question ?

Previous answers:

option_1	50%	5
option_2	20%	2
option_3	10%	1
option_4	20%	2

Total 10

LoRa pod-side

The LoRa pod side don't treat any kind of data, but receive request by the wifi way. So, we have handled it by URL method : we parse the URL base on a given pattern, and match it with local data.

Limitation

At least, we have encountered some technical challenge due to the hardware mainly. This project has been oriented on ESP32 Helltec board which is a modified type of Arduino boards. This board, by his nature, work as a very minimalist station, and is a little bit limited in the complexity that you can put in (memory size, mono-thread, etc...)
But our main problem due to this configuration was the development of the web server. The mobile part of the application was pretty good and, so far, the most advanced we have done, gracefully to the Angular and Ionic framework. On wifi side, unfortunately, we can't work with any kind of framework, and no more with library : the hardware make it too complicated for add library from C++ standard, and due to this limitation, overextended the work amount for doing a minimalist job on the Wifi server part.
So, as a list, we can sum-up it like this :

- The lack of library implemented on the ESP32 Helltec hardware give a ton of overwork for feature already support by standard C++ library (the language supported by the LoRa software).
- A poor memory management.
- A poor documentation on Helltec side (Mainly chinese, and not fully translate).

Other consideration have slow the development, but this is the main one. We have found solution to those matters, for example by using a Raspberry computer for the Wifi part and interconnect it with a LoRa board directly, that could be very easy and reduce the development time by a large amount. Moreover, this problem and our resolution suggestion will reduce the work needed for improvement and long-term support : In fact, you can't deploy any update, fix or improvement easily on ESP32 boards. Moreover, it's really hard to let this project to other people (in the goal of an open sources project) due to the lack of Helltec documentation and the ESP32 limitation. Adopting the Raspberry solution will reach both goal.

Recommendation

We think this project could lead to a really useful framework for special application that require both LoRa and Wifi technologie. The way that we have test it prove this, and the code is not really hard.
But we have to say that, if we still use ESP32 as main material, the project will fail, due to all the reasons expose in the Limitation section. The fact that the framework will be use for really particular application, in few sector, makes it hard for create a group capable to work on and improve it. If we add all the limitation due to the ESP32 hardware, we think that it will be a complete failure.
So, our suggestion will be to change the way that the pod is hosted, by changing the hardware or find a particular library that can be integrated in. We don't have find any solution other than changing the ESP32 boards, but maybe it could be possible to find one, by mailing a request to Helltec developers.