Politecnico di Torino

Projects and Laboratory on Communication Systems
Agriculture Monitoring System

Master degree in Electrical and Computer Engineering

Authors: Group 27

Mattia Cara S254422, Christian Ulrich Fotso Hondjie S240284, Mingyang Zhang S242107

Contents

CHAPTER 1

Goal of the Sensor

Our project belongs to the category of humidity and temperature sensors and it presents one additional UV sensor.The goal of our work is to retrieve some environmental parameter coming from farms (they can be of any kind) in order to understand which relations are present among them and to enhance the productivity. Those data will help farmers in two different aspects:

1. Give a description in real time about the conditions of the culture, in order to intervene in case of emergency.Thus to recover as fast as possible a critical condition.

2. Help farmers to evaluate in a more precise fashion which it will be the final yield, providing previsions about quantity and quality of the product. Of course before reaching an accurate level of precision about these statistics it is required to deploy a big number of sensors and collect a really large number of data (this will be also made possible thanks to the help of big data analysis). Moreover our measures could have been improved with data such as precipitations or wind speed in order to depict a more complete scenario.

CHAPTER 2

Main features

The software we have developed is using a multi-thread approach so that each one of these is going to handle one specific routine.The threads description are in the sections below. Besides this aspect it is worthwhile to explain the role of the set up file that we have designed. Inside the whole program we use two different counters to name files inside the SD card; one counter for measures having a wrong timestamp (a timestamp that comes from the board that has not been synchronized yet with the time server or whenever a synchronization fails, after this failure we are not sure anymore that the internal timer is still correct) and another one for keeping track of files that need to be sent. This set up file (a .txt file) stores these two different numbers and it is used by the board for this main reason: it may happen that the board is turned off and because of this we lose every variable. At the restart if this file is present inside the SD card our board is able to recognize which was the last configuration just before losing power and continue from it.

2.1 Measure Thread

This thread is responsible of reading data coming from sensors every two minutes (we are actually going to use a smaller lapse of time during the presentation). Both temperature and humidity are coming from sensor SI70, while the UV index from GUVA-S12SD .Once these measures have been retrieved we, first of all, save them on the SD card, by doing this operation we are sure that if for any reason the board loses power, as soon as it comes back, data will be still available. After we try to publish three messages, containing the three different values to the broker so that data keep coming out from our board constantly, meaning that everything is fine. If this publish event succeeds correspondent files previously stored in the SD card are deleted; instead if this latter event fails all the files will be kept inside the memory. Every time we read values from sensor we perform a simple verification:

1. Data must be in the range specified by the constructor of the sensor;
2. The value of the sensor that indicates a failure must be false;
3. Values must be different from the previous one (according to thresholds).

For the temperature and humidity sensor, we used I2C protocol to communicate with; while for the UV sensor that is analog we used the 'A' socket which is dedicated to analog interface.

2.2 Synchronization Thread

Values coming from our sensors, in order to have a meaning, must be matched with a timestamp that is assigned every reading operation. Synchronization thread is in charge of keeping updated the internal time of the board, sending periodically a request to a time server. To distinguish files with a wrong timestampsmps with respect to ones with a correct time we map this information in the file name: every file name that starts with wrong must be corrected in the future before being sent, while files starting with correct are ready to be sent. During our first test about this functionality we set this thread to wake up every 15 seconds but this was leading to a block of the board. After many tests, we have supposed, also thanks to the help of prof. Albertengo, that the request to time server is blocking and the fact that we sent too many requests in a short time may have been seen like a possible attack towards the server that decides to not reply anymore. To solve this problem we are now using a thread that wakes up every 6 hours: this doesnt represent a problem to our project because we have tested that even in this lapse of time the delay accumulated by the board is close to zero. Anyway we strongly suggest, during a production phase, to consider using a different library for this operation (documentation provided about TimeService class is almost absent so we werent even able to understand the reason of what was happening) or to implement a watchdog that will restart the board in case of a blocking situation. Every time the timer is changed an event is raised and we wrote a handler of this event to change all the files stored in memory with the wrong label:

1. We save the time before the synchronization starts;
2. We perform the synchronization;
3. We evaluate the difference between the timestamps written into the file and the one we saved in step 1. That difference is used to calculate the new correct timestamp that will be written into the file having this time the label "correct".

2.3 Publish Thread

In the measure thread we just simply try to publish our JSON files, in case of failure we do not perform any operation to recover it. For this reason we have designed another thread that is in charge of sending every file associated with the label correct that is still present in the memory or the buffer. This solution is also fundamental whenever for a certain period of time the internet connection is not available, so that as soon it comes back we are able to send everything. Besides these aspects its also useful every time a file with the wrong label is corrected: indeed wrong JSON are never sent to the database, waiting for a future adjustment.

2.4 Screen Thread

During our development phase we have used a screen to display information about measures that have been read, the timer of the board and the internet status. This tool have been really useful to understand what was happening into the sensors and it may be still helpful for the final user to know their status without using a device to display the web interface we developed.

2.5 Broker and AWS

Since AWS does not support TLS1.1 which is the most recent version supported by our board, we needed to use a broker. When the board needs to send measures to the database, first of all they are sent to a broker based on a EC2 server running Mosquitto Service. There are certificates and keys

generated by AWS IoT so that AWS will recognize it as the IoT device (our board). JSON messages arriving AWS cloud will trigger an event which tell AWS to insert them into the DynamoDB so we can query it and show it later.

2.6 Clear Index Thread

In this simple thread we checked that the content of the SD card, if only one file is present(the setup file) we reset our indexes, so they don't grow indefinitely.

CHAPTER 3

Other Features

3.1 Measure On Demand

We have added to the board a button that allows the user to perform an instant measure. We wrote a method that is triggered when this is pressed and it executes a reading operation on every sensor. This measure is shown on the screen and it is published on the database, if this publish event fails these measures are getting stored into the memory and whenever the publish thread executes its work all of them will correctly be sent to the database. Due to the fact that the pressure operation is not precise enough we added a control so that the event can be raised only once in 5sec. This also does not allow the user to send multiple measurements in a short time.

3.2 Unmounted Flash Memory Mode

We decided to handle a possible situation where the memory is unmounted in order to be able to still work even if with some limitations. This may be really useful if in a future a system of error messaging will be implemented: every time the sensor encounters a major failure it may send a message so that it is possible to recover the emergency as fast as possible. Still there will be a lapse of time between the report of the malfunction and the actual intervention to solve it and this is when our unmounted memory mode is helpful. Instead of saving on the SD card (or in the real case another type of memory) all the measures, we are going to store them in a local array that will be used as a container. Whenever the board is in this mode we drop every measure that has a wrong label and all the publish event are going to read strings from the local array.

3.3 Web Interface

The web interface has been intended to display statistics about every board in the course and not only ours. It is composed by two main parts:

1. The first half of this page is dedicated to show on a map where the board is. There are two different possibilities: on the left there is a list of boards, identified by the device id and on the right all the different type of sensors listed in the JSON configuration file that has been given to us. All the markers that appear on the map are characterized by two main features: a number, that identifies the group, and a color. There are 4 different colors, each one representing a different status of the board, where by status we mean how much time has been passed since the last measure received by the database; these colors are: green (last measure within 3 minutes),

away (last measure within 30 minutes), off (last measure within a day) and dead (last measure received more than a day ago). When the user is displaying sensors based on the category they belong to he has also the possibility to filter them based on this last parameter that we just described: this functionality allows the user to keep under control the general behavior of the sensors; it is instead disabled when the user decides to show sensors based on device id.

2. The second half of the page is instead dedicated to displaying some line charts of the measures performed by the devices. Here the user can choose one board at a time and a set of charts is displayed, one for each sensor present on the board. The total number of measures displayed among all charts is 50 (but this may be modified any time just changing the limit field into the query done to the database). Measures that have a field status different from "OK" are discarded to not spoil the final result. There is also the possibility to show the value of the measure and its timestamps simply moving the pointer close to the interested data.