Intelligent Irrigation System for Low-cost Autonomous Water Control in Small-scale Agriculture

Deliverable D3.6C

Final report on evaluation and KPI assessment in pilots

Responsible Editor:	IRD
Contributors:	UPPA
Document Reference:	INTEL-IRRIS D3.6c
Distribution:	Public
Version:	1.1
Date:	May 2024

Contributors table

DOCUMENT SECTION	AUTHOR(S)
SECTION 1	C. Pham (UPPA)
SECTION 2	G. Gaillard (UPPA) & J.F. Printanier (IRD)

Document revision history

Version	Date	Changes
V1.1	May 2nd, 2024	Public release
V1.0	Apr 15st, 2024	First draft version for internal approval
V0.1	Apr 4th, 2024	First draft

Executive Summary

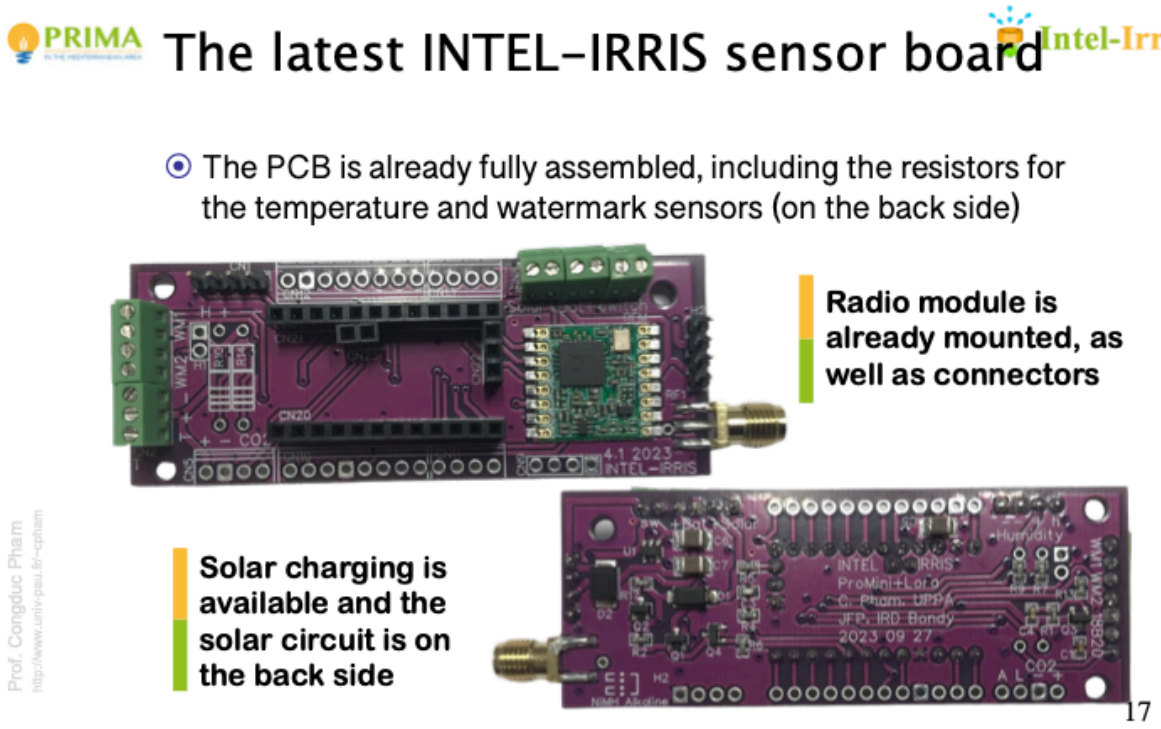
Document D3.6c “Final report on evaluation and KPI assessment in pilots” extends D3.6b “Second report on evaluation and KPI assessment in pilots” with details on the energy consumption of the soil devices based on the IRD PCBA v4.1 which is the hardware platform used by default in the starter-kit v3.

Table of Contents

1. Introduction..... 5
2. Reviewing power provisioning with primary batteries..... 6
2.1. Introduction and context..... 6
2.1.1. Deployment in Settat 2024 faced battery issues..... 6
2.1.2. Power consumption study was with PCBv2 capacitive..... 6
2.1.3. Searching for a software issue..... 7
2.1.3.1. A recent change in powering the DS18B20..... 7
2.1.3.2. A possible mismatch of pin mode setup for sensors..... 7
2.1.4. A debug protocol to identify and tackle the power draining problem..... 7
2.2. Test settings..... 8
2.3. Description of 4 experiments, with 4 hardware & software settings..... 9
2.3.1. Middle term scheme: 2WT with primary batteries and a FTDI32 with the VCC pin disconnected..... 9
2.3.2. Simple fast test scheme: capacitive + FTDI32..... 9
2.3.3. Secured current draw measurement scheme: with a “peak-eater” protecting capacitor..... 9
2.3.4. Second middle term scheme, without a laptop monitoring..... 10
2.4. Monitoring: tuning the Arduino code..... 10
2.4.1. Monitoring the activity duration for each sensor..... 10
2.4.2. CPU-reported Voltages..... 11
2.4.3. Connect the FTDI32 to the RPi’s USB port to monitor the serial output..... 12
2.4.4. Connect the WaziGate to a smartphone-shared Wi-Fi..... 13
2.5. Preliminary validation tests with a multimeter..... 13
2.5.1. Voltage values measurement test..... 15
2.5.2. Current values measurement tests..... 15
2.5.2.1. PCB without Arduino..... 15
2.5.2.2. PCB with Arduino..... 16
2.5.2.3. PCB with Arduino and secured scheme with a capacitor..... 17
2.6. Results using data analysis..... 17
2.6.1. Voltage decreases normally..... 17
2.6.1.1. Experiment with FTDI32..... 17
2.6.1.2. Experiment with radio monitoring only..... 18
2.6.2. Sensors times of activity..... 20
2.6.2.1. Temperature sensor’s update time..... 20
2.6.2.2. Watermarks update time..... 21
2.6.2.3. Capacitive update time..... 21
2.6.3. Estimating the lifetime of a device..... 22
2.7. Conclusions, issues, limits..... 24

1. INTRODUCTION

This final deliverable on “evaluation and KPI assessment in pilots” will investigate in detail the energy consumption of the soil devices based on the IRD PCBA v4.1 which is the hardware platform used by default in the starter-kit v3. Readers can refer to D1.2c “Low-cost sensor generic platforms for connected irrigation system – v3” to have more detail on the IRD PCBA v4.1.



2. REVIEWING POWER PROVISIONING WITH PRIMARY BATTERIES

2.1. Introduction and context

We realized this study in March and April 2024, because of two reasons: first, our Moroccan partners in Settat had detected an issue regarding powering the devices with alkaline primary batteries; and secondly, we intended to explain, generalize and share the internal study made by IRD in early 2023: “Exp_10 INTEL-IRRIS Starter-kit Soil sensor et SEN0308 : alimentation par piles”. In particular, we aim at providing an estimate of lifetime for the devices running with typical heavy duty primary batteries.

2.1.1. Deployment in Settat 2024 faced battery issues

Some devices brought and set up in February 2024, with the final PCBA and powered with alkaline batteries (Duracell), were deployed on the field. 2 main issues were raised:

(1) Feedback from Assia (INRA CRRA Settat)

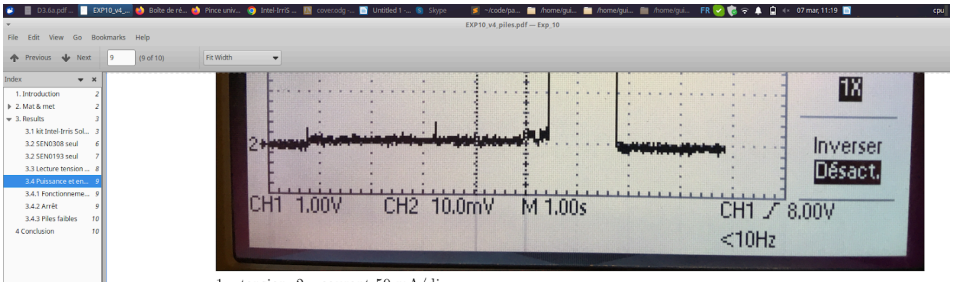
In a field experiment with 4 devices and a gateway, 2 of the 4 devices are not "seen by" the gateway (their data is not collected) until their reset button is pressed. After 24h-48h 2 devices stop being seen (no data on the gateway). Last battery voltage received 2.9V.

(2) Feedback from Abdellah (INRA CRRA Settat)

Issues detected for a total of 4 devices out of 8: the batteries dysfunction (are drained out) after a few hours or days.

2.1.2. Power consumption study was with PCBv2 capacitive

An internal study by IRD was conducted in laboratory conditions with a capacitive device with PCBv2. Among other contents, the study brings the following table:



1= tension, 2= courant 50 mA/div

Table 1: Consommation

Etat	Duree_s	Courant_mA	Puissance_mW	Energie_µWh
Arduino boot	2.7	5	15	11
Start	1.3	5	15	6
Capteur	0.68	8	24	5
Radio Lora	1.32	115	350	128
Veille	3600	0.0047	0.014	14

24 h 3.8 mWh

1 an 1.4 Wh (500 mAh)

We intend to provide and detail the steps to obtain a more complete version of this table.

2.1.3. Searching for a software issue

We tried to explain the issue faced at Settat looking for possible causes In the code:

2.1.3.1. A recent change in powering the DS18B20

The main difference since a batch of previous tests on the new PCBA was due to a commit we made while preparing devices in February (Settat):

https://github.com/CongducPham/PRIMA-Intel-IrriS/commit/d81bb10c2c6ce2a4c294a7d81a308cacae3f3749

The commit adjust and reuse (adds) the “power soft start” procedure in the update function of the temperature sensor (DS18B20), for the devices with PCBA (fully assembled), but without solar panel. Indeed, we noticed the temperature values were not collected otherwise.

2.1.3.2. A possible mismatch of pin mode setup for sensors

The so-called “solar circuit”, added by IRD to the PCBA, on its back face, to deal with power supply with a solar panel, and different non primary batteries (rechargeable NiMh, lithium…) uses a Mosfet transistor connected to the analog pin A1 of the Arduino.

The mosfet gets switched down when A1 is set to input mode (normally, intended during the Arduino boot, and when flashing). And here is the possible mismatch:

- In the main INO Arduino file, in the setup(), the piece of code for the temperature sensor (DS18B20.cpp) is called to set A1 in output mode;
- In the main INO Arduino file, in the loop(), the piece of code for the watermark sensor (watermark.cpp) is called to set A1 in input mode between each update (every hour);

Although surprising, these differences do not seem to impact the measured voltage values (see below). We tried to imitate the watermark code and set A1 in input mode between each update of the DS18B20. Measurements did not change. Thus we stopped looking further in this direction.

2.1.4. A debug protocol to identify and tackle the power draining problem

We reproduced the problem using low-cost AA batteries: in 31 hours we experienced a battery drain on a 2WT device. We tried some hardware tests and measurements, with a multimeter in an unstructured manner, that were not conclusive but enabled us to discuss and **synthetize a detailed electrical debug protocol** for these cases of power drain:

1. We should first check whether the DS188B20 (temperature) is not powered up during sleep: after the emission (the transmission LED is ON during about 1.3s, so, after that), compare voltage between Bat+ et CN2+ (temperature sensor VCC);

2. Then we should measure the instantaneous current draw with a µA meter, by adding a big capacitor (10 000 µF) between the multimeter (caliber 200 µA) and the Arduino, and by doing the measurement ONLY AFTER the radio transmission (current draw 110 mA, way beyond the caliber);

3. Then we should measure the batteries' voltage during the radio transmission, because a low current availability can lead to a decrease in voltage that could make the Arduino continuously reboot and rapidly drain the batteries.

4. Finally, we should take off the batteries, and connect a laptop via USB (FTDI32), and collect the debug log (the serial output) during several days to analyze it.

This protocol proposal was then completed by two steps:

5. Measure the current draw of the device, having removed the Arduino from the PCB. In that case, only the radio chipset is powered and active, this should cause a current draw of about 1.5-2.0 mA. NB: with the Arduino, the radio is off during sleep, current draw gets down to around 5 µA.

6. An improvement to step 4: collect the serial logs using batteries and an FTDI32 without connecting its Vcc pin. This way, the actual behavior of the device when supplied power with primary batteries, supposedly causing trouble, is highlighted. NB: maintaining the connection with the FTDI32 draws current, thus reinforcing the problem of battery drain if any.

2.2. Test settings

- A Device with PCBv2 433 MHz with capacitive sensor;

- A Device with PCBAv4.1 868 MHz with 2 watermark sensors, 1 DS18B20 sensor;

- Two Wazigate gateways (2022, 2023): black casing, updated software, one 433 and one 868 MHz.

- Two Ubuntu laptops, a connexion sharing smartphone, an ethernet cable;

- Two FTDI32 (version from chinese manufacturer HWA YEH), one with a 6-pin Female connector, the other with 5 jumper wires, (VCC pin left unconnected).

- 4 Alkaline Duracell Optimum batteries (heavy duty);

- 4 AA low-cost primary batteries;

- 1 multimeter, possibly with hook clips;

- an electronic breadboard and jumper wires;

- a 10000 µF capacitor, a 10 kΩ resistor, a press button switch;

- A lab power supply, but the primary batteries otherwise.

2.3. Description of 4 experiments, with 4 hardware & software settings

After the first short test using low-cost AA batteries on the 2WT, we conducted longer tests.

2.3.1. Middle term scheme: 2WT with primary batteries and a FTDI32 with the VCC pin disconnected

Duration: March 25th - April 8th;

- Device 2WT 868 MHz PCBA with two heavy duty Alkaline, and connected to the FTDI32 without VCC;

- Monitorization of the serial output via USB and FTDI32 on a laptop;

- Hourly temperatures collected on Wazigate.

2.3.2. Simple fast test scheme: capacitive + FTDI32

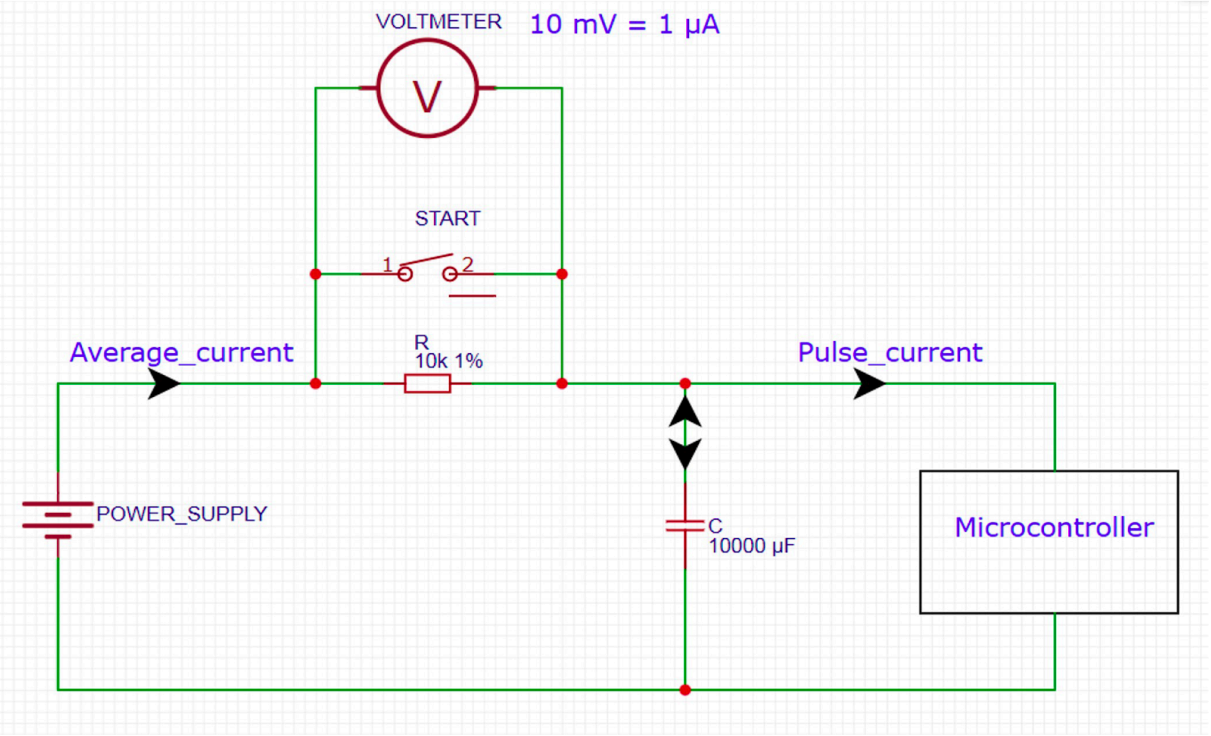
Duration: various tests, max 3h.

- Device capacitive 433 MHz PCBv2 powered by FTDI32, without batteries

- Monitorization of the serial output via USB and FTDI32 on a laptop;

- Arduino code in debug mode #define TEST_LOW_BAT: one sensor cycle every minute approximately.

2.3.3. Secured current draw measurement scheme: with a “peak-eater” protecting capacitor



This setup has been built in Settat with the following modifications/specificities:

- Use for the power supply of the Duracell alkaline batteries that were used on the field deployment when the issue was detected;
- Use of a set of capacitors in parallel summing a total capacitance of 10000 µF.

2.3.4. Second middle term scheme, without a laptop monitoring

In the first middle term scheme, we faced the issue of constantly checking the state of the laptop (battery, sleep mode, serial connection). Eventually, it rebooted twice without a particular reason, overnight, thus we lost 2 periods of data. Besides, the presence of the FTDI32 draws current from the Device's batteries, and impacts the experiment.

- Device 2WT 868 MHz without FTDI32, on new primary heavy duty Alkaline batteries;

- Monitorization of voltage values on the Wazigate (radio output);

- Arduino code in debug mode #define TEST_LOW_BAT: one sensor cycle **every 10 minutes** approximately. This period was chosen to increase the power demand of the device, and accelerate the data extraction to estimate the lifetime. Still, we chose 10 instead of 1 minute in order to be sure to maintain a sleep period that causes a similar impact on batteries and components.

2.4. Monitoring: tuning the Arduino code

2.4.1. Monitoring the activity duration for each sensor

In order to measure this, we first thought that timestamping the serial monitor would be enough. We use TIO v2.8 (https://github.com/tio/tio) as serial tool to timestamp the serial logs, e.g. using:

~$ tio -b 38400 -l --log-file /home/guigui/tempuino.log --timestamp --timestamp-format iso8601 /dev/ttyUSB1

The **first idea** was to tag the update() function of the sensors with two serial prints ("begins", and "ends"). This was implemented on March 29th, as follows, on the 2WT device.

Extraction:

```
152  [2024-03-31T20:16:15.973] CRC,43FA
153  [2024-03-31T20:16:15.973] LoRa pkt size 37
154  [2024-03-31T20:16:15.973] LoRa pkt seq 97
155  [2024-03-31T20:16:15.988] LoRa Sent in 1968
156  [2024-03-31T20:16:15.988] Switch to power saving mode
157  [2024-03-31T21:20:34.355] Wake from power saving mode
158  [2024-03-31T21:20:34.355] BATTERY-->3.41 | 3.34
159  [2024-03-31T21:20:34.369] update DS18B20 begins
160  [2024-03-31T21:20:35.389] update DS18B20 ends
161  [2024-03-31T21:20:35.389] update watermark begins
162  [2024-03-31T21:20:35.404] 32760
163  [2024-03-31T21:20:35.404] update watermark ends
164  [2024-03-31T21:20:35.404] update watermark begins
165  [2024-03-31T21:20:35.420] 32760
166  [2024-03-31T21:20:35.420] update watermark ends
167  [2024-03-31T21:20:35.420] batvoltage
168  [2024-03-31T21:20:35.420] 3.34
169  [2024-03-31T21:20:35.420] batvoltage_was
170  [2024-03-31T21:20:35.436] Sending \!WM1/3276.00/CB1/255.00/WM2/3276.00/CB2/255.00/ST/18.87
171  [2024-03-31T21:20:35.452] Real payload size is 56
172  [2024-03-31T21:20:35.452] use LPP format for transmission to gateway
173  [2024-03-31T21:20:35.468] end-device uses native LoRaWAN packet format
174
175  [2024-03-31T21:20:35.484] plain payload hex
```

We then used a python script to measure the update() time by computing the time difference between the “begins” and “ends” timestamps.

We suspected the serial interface would cause some time variations, impairing the delay accuracy. **Second idea** was then to compute the time difference directly on the Arduino, using the millis() function. Implemented on April 3rd as follows:

```
86
87  double watermark::get_value()
88  {
89    long startupdate = millis();
90    Serial.println("update watermark begins");
91    update_data();
92    Serial.print("update watermark ends:");
93    Serial.println(millis()-startupdate);
94    return (get_data());
95  }
96
```

NB: here we are looking for an upper bound of the update() time. Thus, the inclusion of the two serial.println() calls in the calculation of difference is not impairing.

2.4.2. CPU-reported Voltages

In order to measure the voltage without adding components and complexity to the PCB, in Intel-IrriS the Arduino code has taken advantage and inspiration from the Arduino_VCC library (https://github.com/Yveaux/arduino_vcc).

During the radio transmission, the current demand is so high that the voltage supply decreases. Before mid-April, the code only provided a measurement of voltage value during transmission when using a solar panel. Indeed in that case, this value determines whether the device has enough power supply to work, or needs to wait for a solar charge.

We modified the code to use the Arduino_VCC during transmission for the cases without a solar panel.

https://github.com/CongducPham/PRIMA-Intel-IrriS/pull/19

This PR merges two previous ones:

- Voltage during tx, code refactoring, nextTransmissionTime #17

In the case of a device that does not use the solar portion of the circuit to manage batteries, e.g. a PCBA with Alkaline batteries, this PR enables to get voltage measurements from the CPU during transmission. These later are printed on serial, and can be sent to the gateway by radio, separately in debug mode (TEST_LOW_BAT), or as the minimum value (default).

Finally, after testing all, we have modified the computation of nextTransmissionTime to better comply with the expected behavior: instead of adding extra time between measurements when low voltage is detected, the nextTransmissionTime is now multiplied by a chosen factor in that case. The transition between short and long intervals, meant to warn the end user beforehand, is now implemented in a simpler way.

- impose a sleep period after 3 consecutive reboots #18

This PR implements a counter of reboots during TX. It is stored on EEPROM and reset to zero when any TX goes well. Otherwise, the measure_and_send() function is prevented once.

We implemented this merged pull request, i.e. the new version of the code including the changes, using scheme 4: Second middle term scheme. No need for a laptop anymore, we will be able to extract the data directly from the GW.

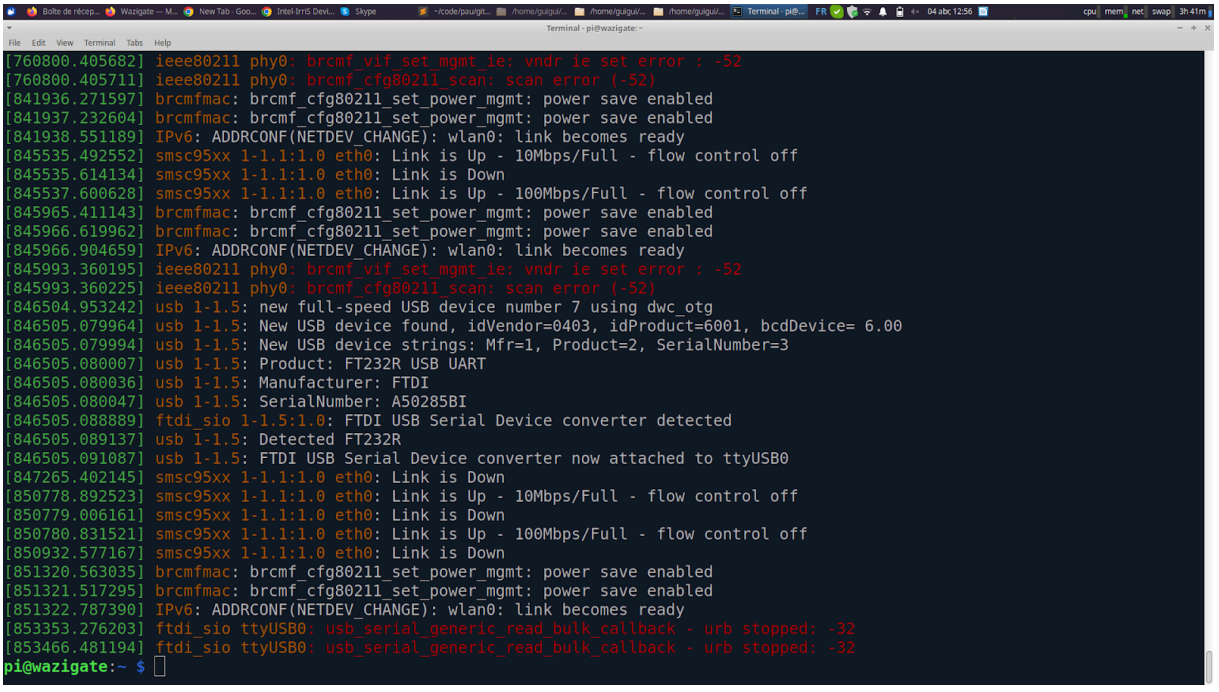
NB: Each Arduino board would report a different voltage value in the very same situation. A calibration is normally necessary to match this with voltmeter values, by adjusting VccCorrection in the code.

2.4.3. Connect the FTDI32 to the RPi's USB port to monitor the serial output

The WaziGate's RPi has 4 USB connectors. Instead of using a laptop, we tried as a third idea to collect the serial output of the devices directly connecting the FTDI32 to this USB port.

In order to do that, you need a terminal multiplexer on the RPi, such as Tmux or Screen: this way, you can trigger a serial monitor that would pursue its logging when you close the SSH session on the WaziGate.

But there is a conflict between the FTDI32 (chinese version) and the RPi: the serial connection breaks after some variable time. DMESG journal shows a non-trivial error message, "ftdi_sio_ttyUSB0 [...] urb stopped: -32". So we did not go further there.



2.4.4. Connect the WaziGate to a smartphone-shared Wi-Fi

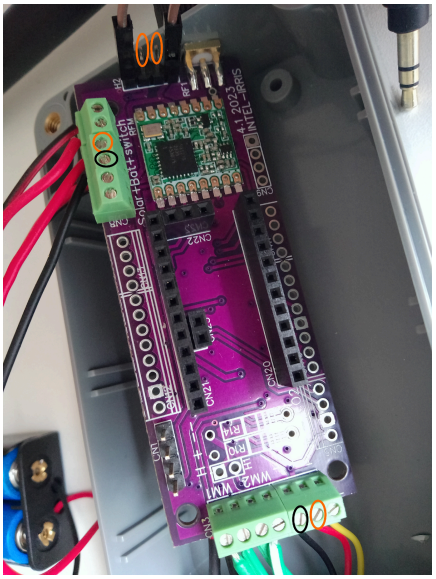
The middle term experiments took place without the presence of a continuous WiFi connection provided by an Access Point. Instead, the Wi-Fi gets available from time to time, in proximity to the personal smartphone used in sharing connection mode.

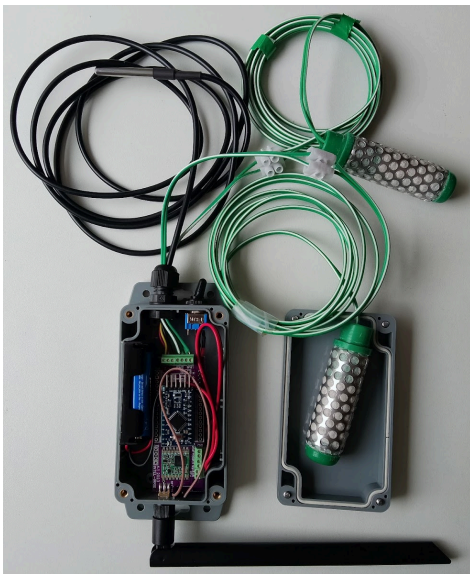
This revealed a quite correct behavior:

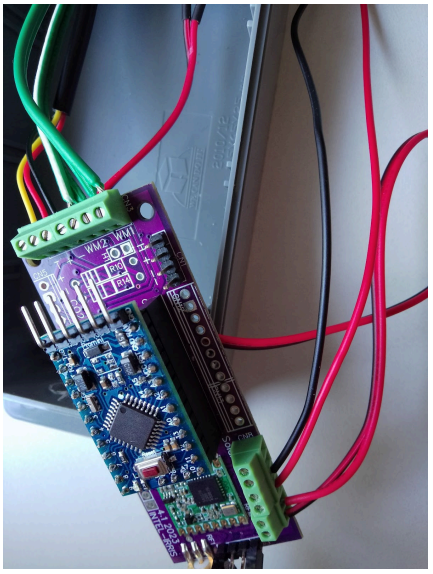
- The reconnection delay is no longer than a few minutes;
- The Wazigate switches easily to AP mode when the connection is not shared anymore;
- The DNS wazigate.local is successfully adapted to reach the Wazigate in AP or Client mode;
- One slight detail though, that we already detected on Settat: accessing the dashboard from the smartphone requires the Wazigate to be in AP mode and the phone's mobile data connection to be off. Otherwise, the web browser on the phone (in this case Android, Realme C31), prioritizes and waits for a response from the Data network instead of the local Wi-Fi network provided by the AP.

2.5. Preliminary validation tests with a multimeter

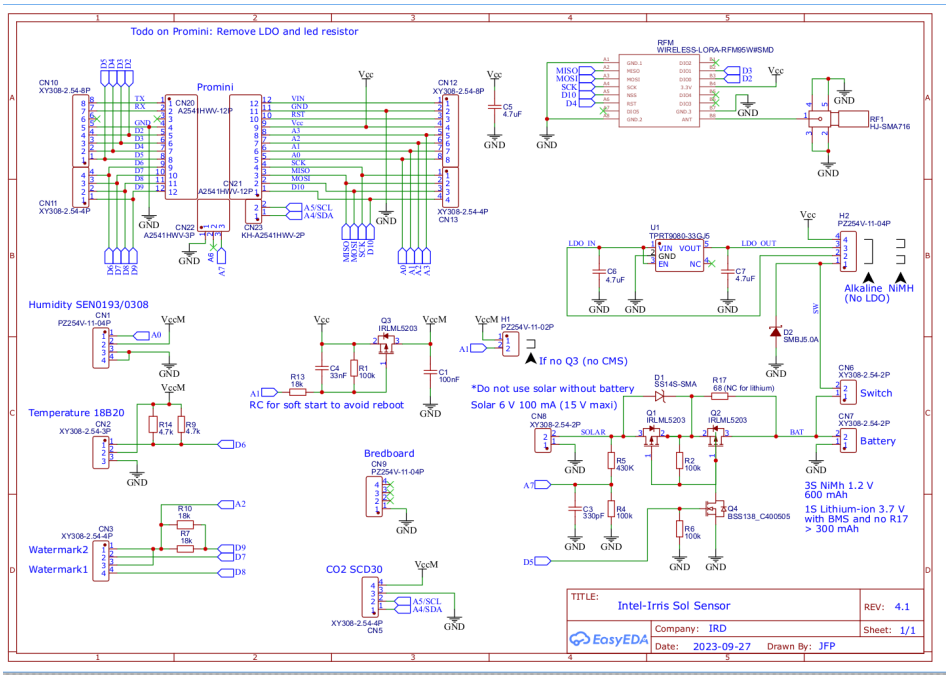
Before entering into longer experiments, the idea here is to validate that the device in our hands has no specific hardware issue, related to the PCB, the connections, the Arduino…







And here is the schematic: (see PDF in github)



2.5.1. Voltage values measurement test

Regarding the first step of our debug protocol:

check whether the DS188B20 (temperature) is not powered up during sleep: after the emission (the transmission LED is ON during about 1.3s, so, after that), compare voltage between Bat+ et CN2+ (temperature sensor VCC);

1. Using new (low-cost AA) batteries or USB powering (via FTDI32), during sleep, the voltage between CN2-2 (temperature +) and Bat+ is static between 2.5 V 2.7V.
2. Other voltage values regarding the temperature sensor:
    a. Without the Moroccan commit: V{t+,t-}: 3.2, V{Vcc,t+}: 0, V{Vcc,t-}: 3.2, V{Vcc,tT}: 0;
    b. With the Moroccan commit: V{t+,t-}: variable around 0, V{Vcc,t+}: static around 2.7, V{Vcc,t-}: 3.2, V{Vcc,tT}: static around 2.7
3. Other voltage values regarding the radio transmission: the voltage drops to **2.86 V** during transmission with low-cost AA batteries

Analysis: 1. and 2.b. seem to indicate Vcc-Vccm(T+) = 0.5V of remaining voltage in the sensors, which would be normal, and causing a neglectable current draw.

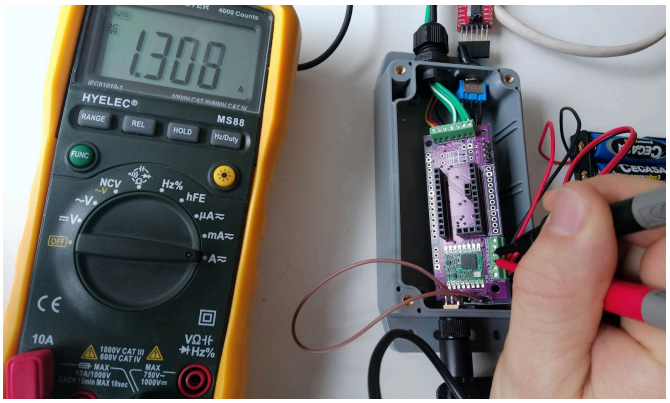
3. seems to show a limit of using low-cost batteries: in the code, the minimum voltage value the software accepts is 2.85 V. This safeguards the 2.74 V value which is a minimum for the Atmega CPU (and the Arduino) to boot.

2.5.2. Current values measurement tests

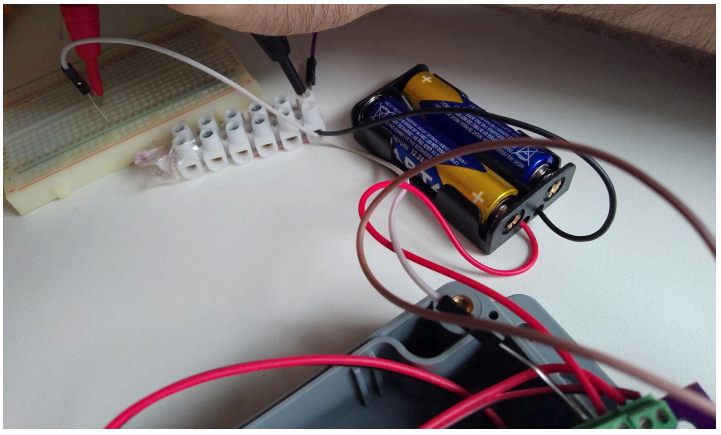
2.5.2.1. PCB without Arduino

In a naive approach, surely but unwillingly ignoring the fundamentals of common and safe use of a multimeter, we first measured here the short-circuit current draw of the low-cost batteries. Using a 10 A caliber during a few moments, we first measured 1.3 A between BAT+ and BAT-. A huge energy consumption that got us excited, “is it the bug?” :).

Although naive, this value seems significantly coherent: 1.3 A at 3 V corresponds in a first approximation to 0.3 V with a demand of 130 mA => during the radio transmission, which has such a demand, the voltage would drop about 0.3 V, reaching the minimum value of 2.7 V.



2.5.2.2. PCB with Arduino



The following week, we continued trying to diagnose the battery outage issue faced with the PCBs.

Assuming the current draw is constant and very little during sleep time, it is possible to use a short caliber of the amperemeter to obtain a precise measure. The difficulty of this measurement is that you need to get in the sleep state before doing it. The internal resistor of the amperemeter in such caliber and/or the switching from one caliber to the other makes the voltage drop and the Arduino reboot.

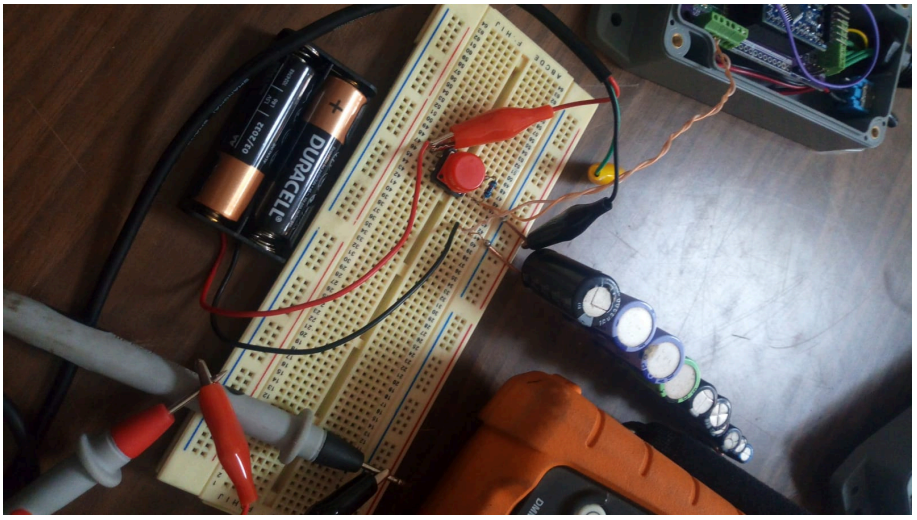
Solution: short-circuit the amperemeter during boot and sensor cycle, then remove the short-circuit after transmission, in order to do the measurement.

Caveat: during the sleep period, the ATMega microcontroller wakes up every 8 seconds with a current draw 1000 times more important. One should not maintain the measurement active more than a few seconds in order to avoid the risk of over-powering the multimeter. Same for the risk of a sudden reboot. NB: It is possible that the capacitor(s) in the “solar circuit” prevent this risk by charging this periodic current.

We measured this way the current draw of the device in sleep mode: 6.3 uA, that is to say, perfectly normal, as expected.

This tends to make us discard the hypothesis of a current leak during sleep state. A relief for the PCBs designers, but then we still need to find the explanation elsewhere.

2.5.2.3. PCB with Arduino and secured scheme with a capacitor



This was tested in Settat, so that we should be able to overcome the caveat of the simple experiment (measurement without capacitor with the CPU waking up every 8s).

Note that you must check the leakage current of the capacitors before connecting the device. That way, you can take them into account in the measurements.

Up to now, no stable result has been achieved that way.

2.6. Results using data analysis

By data analysis, we refer here to the extraction of information from serial output logs, and/or from data gathered by a Wazigate.

2.6.1. Voltage decreases normally

2.6.1.1. Experiment with FTDI32

Here, the logs were collected during the first middle term experiment (see here up) during 14 days.

By filtering out rows including “BATTERY-->”:

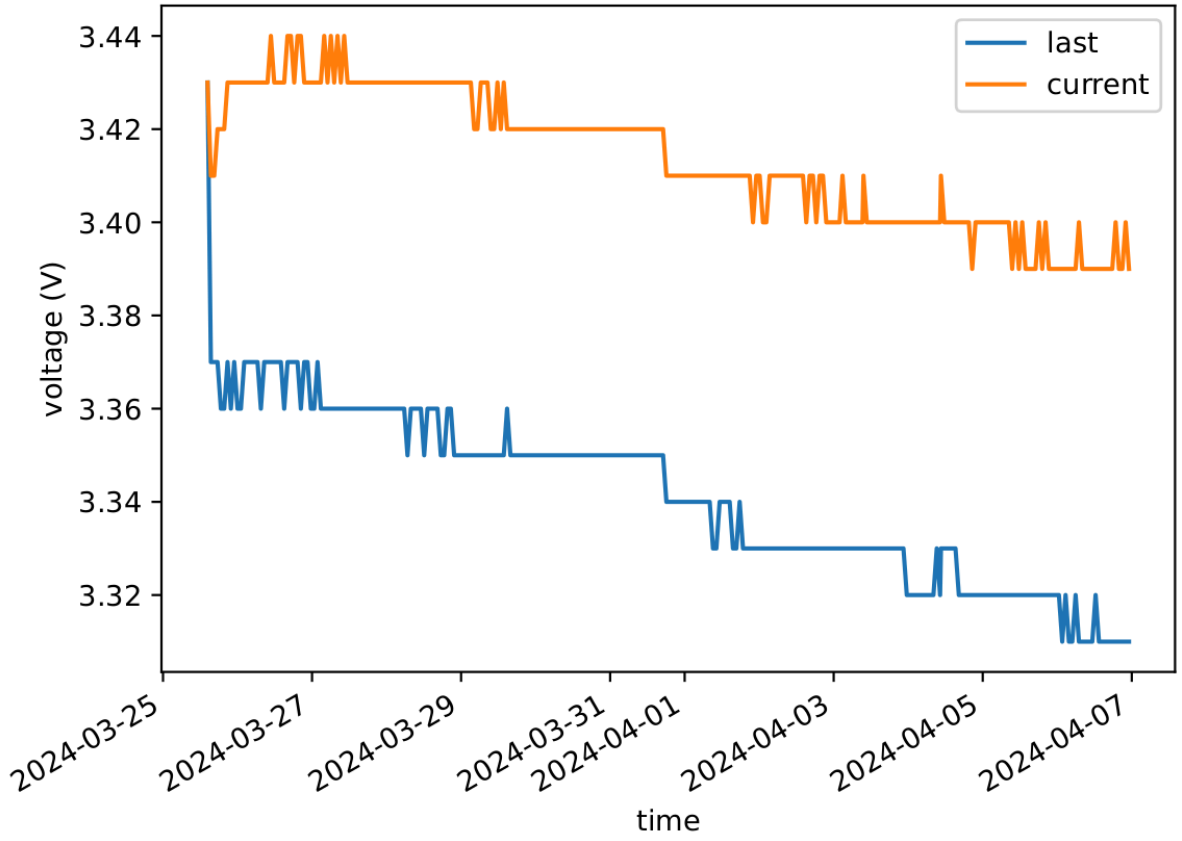
[2024-03-26T00:00:09.157] BATTERY-->3.43 | 3.36

We read and plot a graphical representation of the two voltage values:

BATTERY--> <current > | <last>

Current: Voltage read by the CPU at the beginning of activity (low current demand). NB: it is not a current value, it is a voltage value in V. It is called “current” as opposed to “last”, i.e. previous, anterior, etc. because it is sent over the radio during the very same activity cycle.

Last: Voltage read by the CPU at the end of activity (after transmission) (higher current demand). NB: “last” is measured after transmission, so the value that is sent over radio corresponds to the measurement done during the previous cycle of activity (“last”). For the first transmission after flashing, “last” is set to the value taken by “current”. Since we reprogrammed (reflashed) a couple of times the Arduino during the experiment, we had to remove the first values from the graphical representation, for the sake of clarity.



The voltage decrease is of around 60 mV in 14 days ((3.37 - 3.31) x 1000). Assuming a minimum functional voltage of 2.74 V for “last”, the batteries would work for 147 days:

((3.37 - 2.74) x 1000) / (60/14) = 147

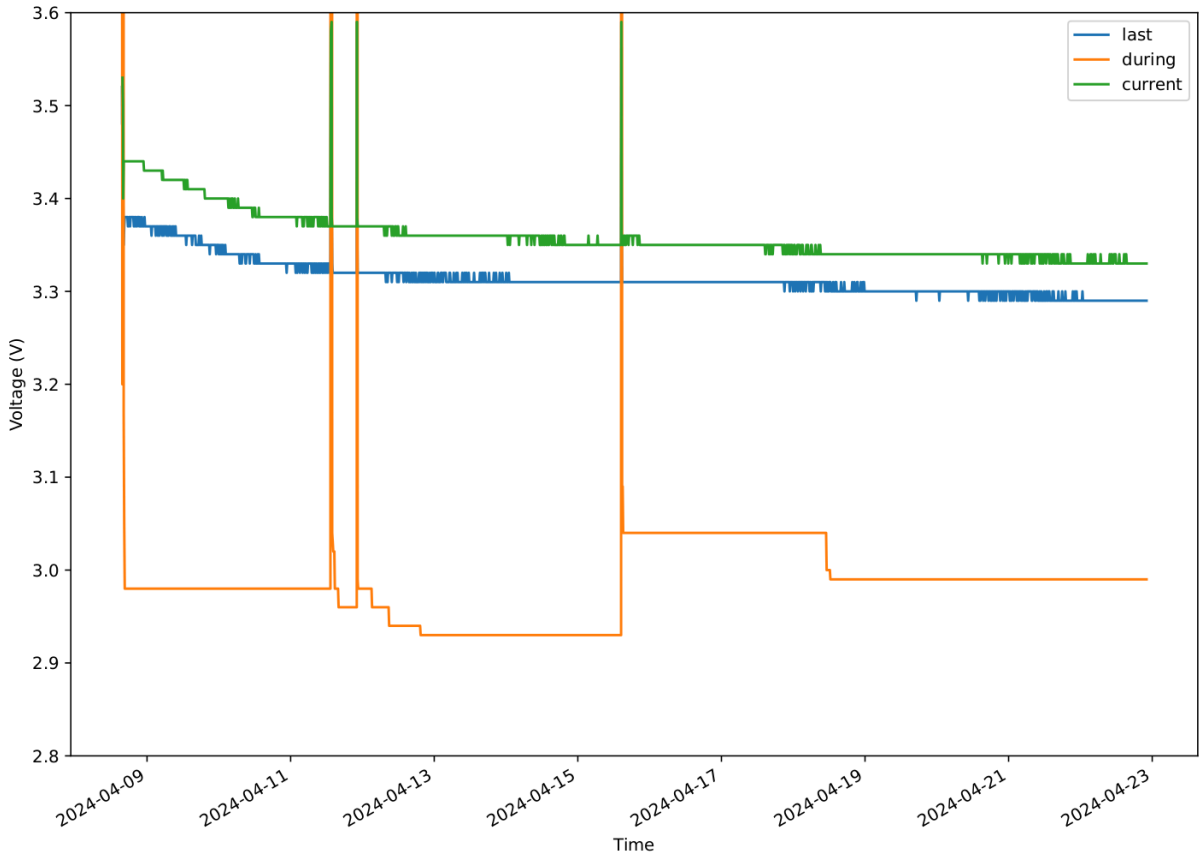
NB: we have seen that this estimate was wrong: a few days after April 8th, the batteries got drained. And at the opposite side, we have examples of experiments working during 14 months, i.e. 420 days. This can be explained for several reasons:

- During transmission, the current demand is higher than during the measurement of “last”. The minimum value is reached earlier;
- In the code, the minimum voltage value is set at 2.85 V, so the normal behavior of the device breaks earlier;
- The FTDI32 is powered by the laptop, but the connection to it draws current from the Arduino, so the voltage supplied decreases faster.

2.6.1.2. Experiment with radio monitoring only

We confirmed (at least as a possible reason) in the first middle-term experiment that the connection to the FTDI32 significantly increases the power consumption.

We now run the other experiment using the radio, a gateway, and a sleep time of 10 minutes. The voltage **during** transmission is now monitored too.



Analysis:

- The voltage decrease is slower without the FTDI32;
- The sleep cycle being reduced to 10 mins, the energy consumption is increased (the power-hungry activity period occurs 6 times more frequently, but it’s not exactly 6 times more energy consuming since the device also draws current during sleep and the sleep period is decreased…);
- The voltage values during transmission decrease very slowly, letting us expect a long life to the batteries;

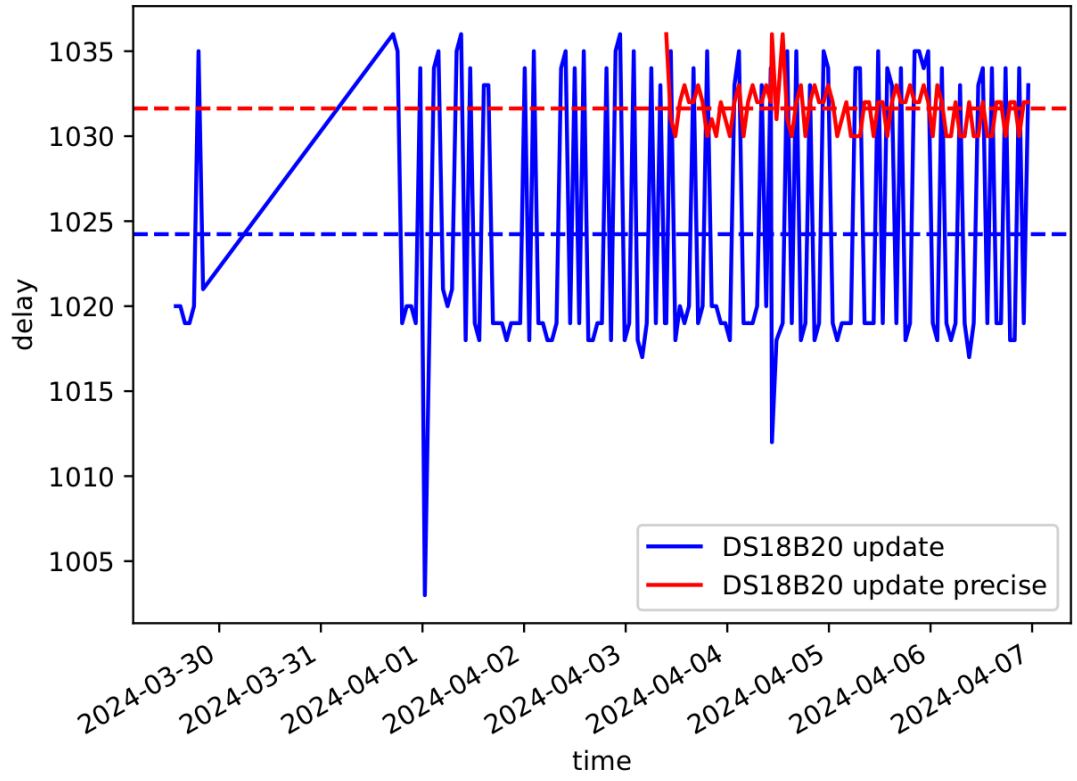
- Surprisingly, the voltage values during transmission even get steady **higher** values after reprogramming the device, as if the batteries were benefiting from the holiday break (time of upload, during which the FTDI32 takes care of powering the device). For instance, we reached 2.93 V on April 13th, but we are still back and steady at 2.99 V only, after reprogramming on April 15th evening, and up to now on 22nd.
- There is still no clear explanation for why the voltage values during tx change (steady state to an increased value) after reflashing the Arduino. It may depend on the battery technology and/or sample, but much more tests on a wider variety of batteries will be needed.

2.6.2. Sensors times of activity

In order to bound the energy consumption during measurement, and assuming the sensors' consumption is the most power-hungry during measurement, these "up" times would help.

2.6.2.1. Temperature sensor's update time

We found about one second (1.032 s) for the DS18B20. NB: this value depends both on hardware and on software choices.

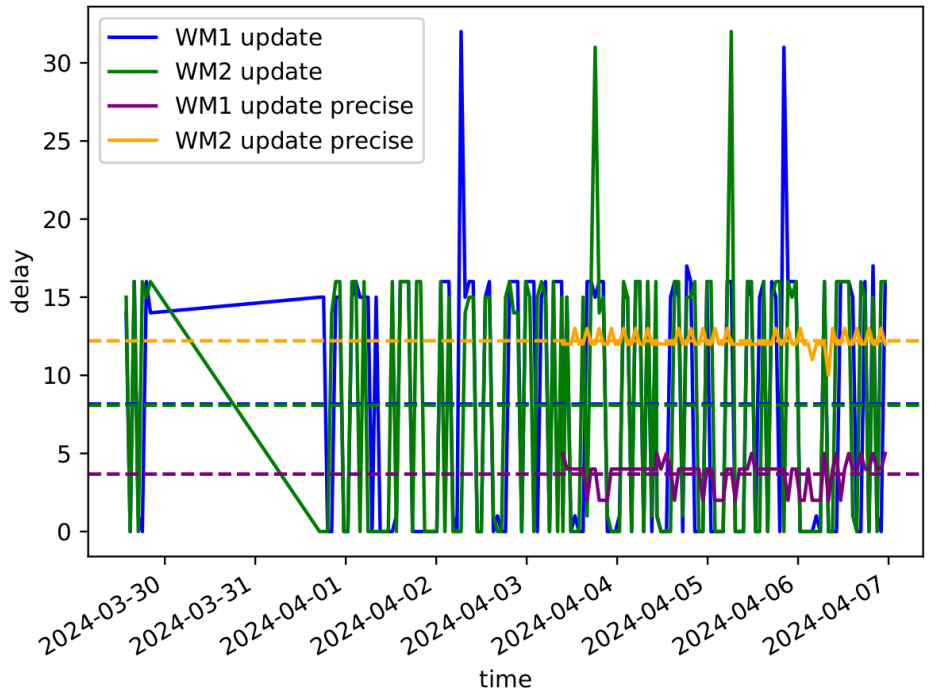


The "Serial readline" (we assume TIO, the serial monitor used here, reads the serial data and bufferize it row per row, before timestamping each buffer) takes a varying time, so the program under-estimates the measured delay in blue (based on these timestamps).

The second approach directly using Arduino's millis() is more precise (red).

NB: the laptop inopportunely rebooted on March 30th.

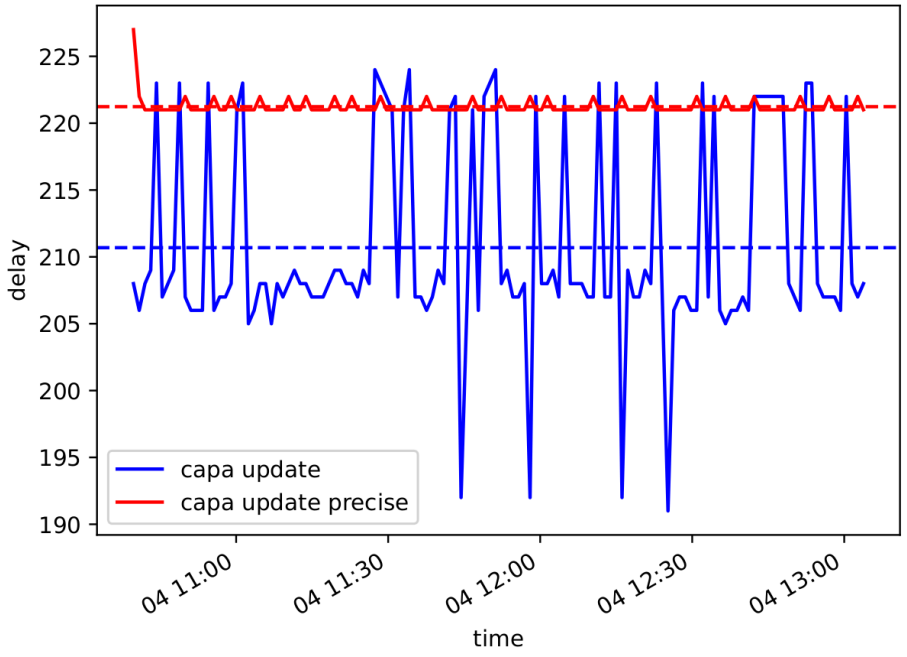
2.6.2.2. Watermarks update time



Same idea about the time precision of the serial output. Here, two watermarks are evaluated.

The major difference with the temperature is that the update time is very short (the mean is around 8 ms).

2.6.2.3. Capacitive update time



For the capacitive sensor update time (same principle), we used the fast simple experiment.

The monitoring code focuses on the “raw analog” update, since the capacitive inherits from this generic piece of code… during 2 hours approx.

The delay found here is around 221 ms. This is not what was considered in the study from IRD. Indeed, the “Capteur” row in the table was meant to represent the full activity of the Device between wake up and transmission (excluded).

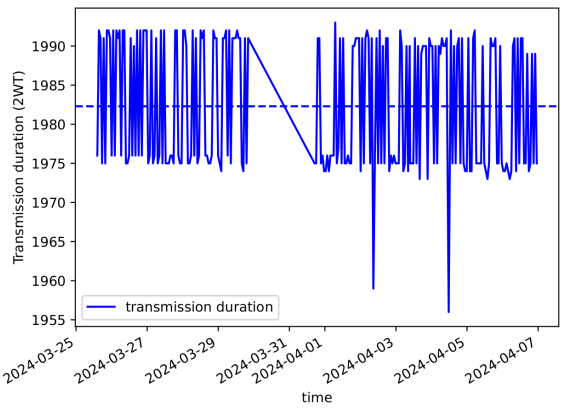
NB: as we said, some delays during update are chosen in the code. The time value will also differ depending on the version of the code (successive improvements of the measurement accuracy and efficiency).

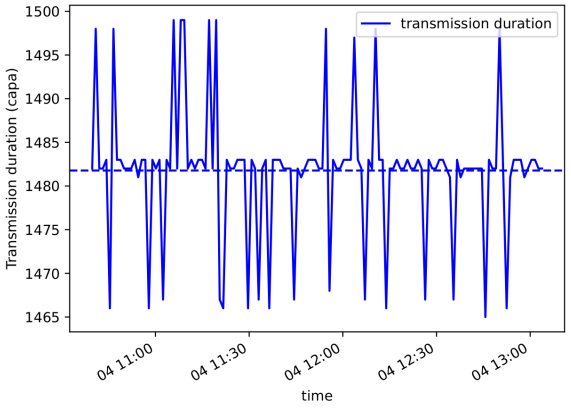
2.6.3. Estimating the lifetime of a device

Let’s take the same idea as previously: let’s consider three states for the Device:

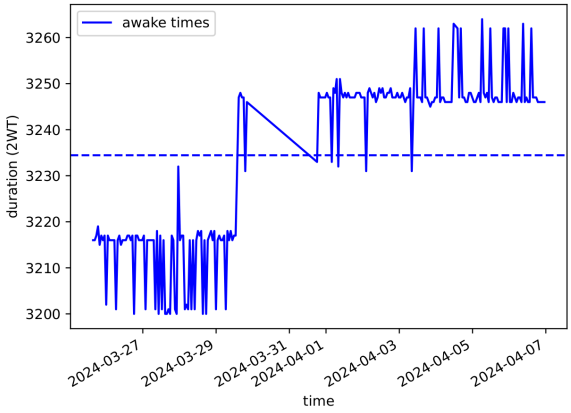
- Sleep;
- Sensor (and other) activity;
- Radio transmission.

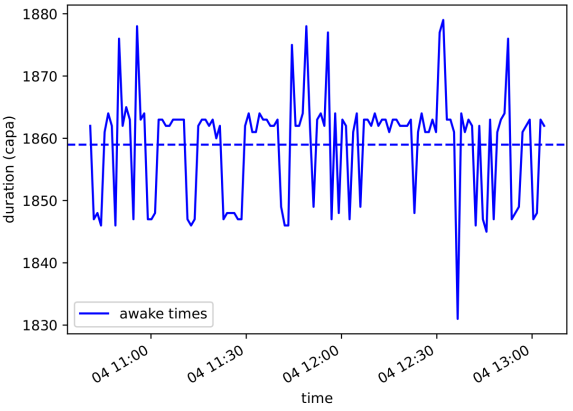
Using the time stamped serial logs as previously, and being aware of their intrinsic imprecision, let’s estimate the **measured** duration of each state:





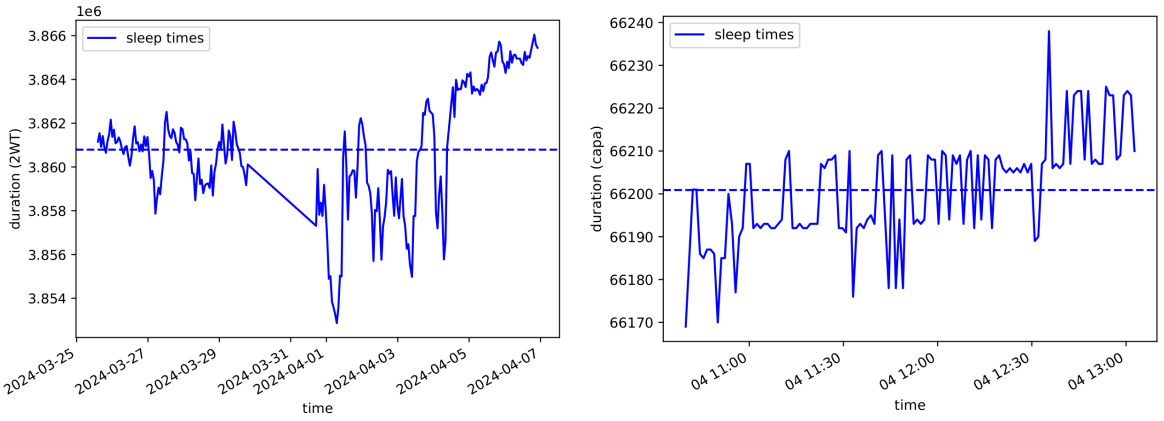
The transmission time seems longer for the PCBA 2WT device, 1982 ms (reported 1968 ms by the Arduino code), than for the PCBv2 capacitive device, 1482 ms (reported 1482 ms by the Arduino code). This difference could be explained by the size of the LoRa messages (21 bytes for the capa, 37 for the 2WT).





The total awake times are around 3235 ms for the 2WT, 1859 ms for the Capacitive device. NB: the increase in duration after March 29th for the 2WT is due to the addition of 4 serial prints. Corrected values would be 3217 ms (2WT) and 1843 (Capacitive, 2 prints).

Finally, sleep times are not exactly what they are coded to be, this is due to the management of the clock drift of the Arduino during sleep state.



Mean sleep time for these two devices are about 3861s for 1 hour expected (2WT), and 66.2s for 1 minute expected (Capacitive). These values tend to increase in time.

Reporting all these values and adding some assumptions:

- Assume a max consumption of 0.01 mA during sleep;
- Assume a max consumption of 8.0 mA during sensor activity;
- Assume a max consumption of 130.0 mA during radio activity;
- Assume the voltage is always 3 V;
- Assume the capacity values of the batteries are provided for exactly the use case of Intel-IrriS devices operation (a mix of (very) high and (very) low power demand in a mix of heterogeneous delays);
- Assume transitions between awake and sleep states are done “time & energy”-less.

=> **We obtain:**

	duration	current	power	energy
	s	mA	mW	µWh
Sensing activities – 2WT	1.25	8.00	24.00	8.33
Sensing activities -- Capa	0.38	8.00	24.00	2.51
Radio Lora Tx – 2WT	1.97	130.00	390.00	213.42
Radio Lora Tx – Capa	1.48	130.00	390.00	160.33
Low power (sleep)	3861	0.01	0.03	32.18
Short low power (sleep 10min)	643	0.01	0.03	5.36

	Capacitive Device (C)	2 Wm + Temp. Device (2WT) (1h)	2 Wm + Temp. Device (2WT) (10’)	
1 cycle	195.02	253.92	227.10	**µWh**
24h	4.36	5.68	30.36	**mWh**
1 year	1.59	2.07	11.09	**Wh**
energy for 1 year at 3V	531.08	691.22	3,696.78	**mAh**
days with 2*100mAh AA batteries	137.55	105.68	19.76	**days**
days with 2*200mAh AA batteries	275.10	211.37	39.52	**days**

Analysis: this is a bit more pessimistic than in the IRD study, however, it is taken with worst-case values. NB: these estimations are subject to a lot of factors not considered here, e.g. but not limited to:

- External temperature;
- Variability among batteries;
- Variability among Arduinos;
- Etc.

2.7. Conclusions, issues, limits

Concluding remarks:

- With heavy duty batteries, the PCBA we tested does not drain its batteries faster than expected, and all the measurements done are explained;
- The study allowed for great improvements of reboot cascade management, and of debug TEST_LOW_BAT mode with radio transmissions of voltage values during transmission;
- The estimates are pessimistic but not really trustworthy due to the assumptions one must take, and to the factors impossible to include;
- Nevertheless, the estimation highlights the fact that the radio transmission is longer for a 2WT in debug mode than for a simple C type device: the power consumption difference is significant;
- A debug protocol has been provided and partially verified;
- As being a yet another debug/preparation test, a possible verification of batteries+Arduino could be to program the Arduino in TEST_LOW_BAT mode and collect 10-12 minutes of data; this way, the voltage values before, after, and during transmission **can be verified** (the voltage during transmission should be somehow upper than 2.9 V and steady during the first minutes) **before a field deployment.**

Project Co-ordinator Contact

Pr. Congduc Pham

University of Pau

Avenue de l'Université

64000 PAU

FRANCE

Email: Congduc.Pham@univ-pau.fr

ACKNOWLEDGEMENT

This document has been produced in the context of the PRIMA INTEL-IRRIS project. The INTEL-IRRIS project consortium would like to acknowledge that the research leading to these results has received funding from the European Union through the PRIMA program.