CONSOMMATION D'ENERGIE DES SOCS RAPPORT TP 2

1 INTRODUCTION

Le but de ce TP est de mesurer la consommation des objets connectés et de concevoir une application intégrant la gestion de la consommation. On s'intéressera en particulier à la technologie radio LoRa très utilisée dans le monde de l'IoT. On proposera également un scenario applicatif d'objet connecté en LoRa en mettant en œuvre un prototype à l'aide la plateforme UCA-LoRa.

2 ETUDE DE LA CONSOMATION DE LA PLATEFORME UCA-LORA

2.1 TECHNOLOGIE DE COMMUNICATION LORA

Le LoRa est une technologie de communication de longue portée, de bas débit et de basse consommation. Le LoRa est utilisé pour la communication d'objets connectés pour créer des environnements intelligents comme les « smart cities » .

Pour mesurer la consommation d'une émission LoRa, nous utilisons la plateforme UCA-LoRa. Le processeur est un ATMega328p que nous programmons avec l'IDE Arduino. On programme une carte UCA-LoRa avec un programme d'émission LoRa (LoRaSender) et une autre carte avec un programme de réception LoRa (LoRaReceiver).

LoRaSender

```
#include <SPI.h>
#include <LoRa.h>

#define LORA_KEY "QR_pkg-"

int counter = 0;

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("LoRa Sender");

  if (!LoRa.begin(866E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  LoRa.setTxPower(1,1);
  LoRa.setSpreadingFactor(7);
  LoRa.setPreambleLength(8);

}

void loop() {
  Serial.print("Sending packet");
  Serial.println(counter)

  LoRa.beginPacket();
    LoRa.print(LORA_KEY);
    LoRa.println(counter);
    LoRa.endPacket();
    counter++;

  delay(1000);
}
```

Figure 1- Programme LoRaSender

```
#define LORA_KEY_SIZE 7
#define LORA_KEY "QR_pkg-"

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("LoRa Receiver");

  if (!LoRa.begin(866E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  LoRa.setSpreadingFactor(7);
  LoRa.setPreambleLength(8);
}

void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {

    String read_data;
    while (LoRa.available()) {
      read_data +=(char)LoRa.read();
    }
    if(read_data.substring(0,LORA_KEY_SIZE) == LORA_KEY){
      Serial.print("Packet received : ");
      Serial.println(read_data.substring(LORA_KEY_SIZE));
    }
  }
}
```

Figure 2- Programme LoRaReceiver

Afin de « faire le tri » entre tous les messages transmis pendant le TP, nous avons ajouté un identifiant « LORA_KEY ». Cet identifiant est envoyé au début du message pour signaler qu'il est envoyé par plateforme de notre groupe. Le récepteur n'affiche le message que si l'identifiant correspond à la clé LoRa choisie.

Figure 3- Debug LoRaSender

Figure 4- Debug LoRaReceiver

La consommation d'une émission LoRa dépend principalement de deux facteurs configurables :

- Le gain à l'entrée de l'émetteur (TxPower) qui détermine l'amplification du signal émis. Et TxPower plus grand est synonyme d'une plus grande consommation
- Le Spreading Factor (SF), qui détermine la redondance de l'information envoyée. Un SF plus grand s'accompagne d'une temps « On air » plus grand et donc d'une plus grosse consommation.

Les cartes sont alimentées en utilisant un câble USB. Pour mesurer la consommation, nous utilisons le module USB UM25C qui nous permet de récupérer le rapport de consommation sous format csv. Nous mesurons l'impact du TxPower et du SF sur la consommation.

2.2 INLFUENCE DU GAIN DE L'EMETTEUR (TXPOWER)

On mesure d'abord l'impact du TxPower sur la consommation. On configure la puce LoRa pour utiliser un SF de 12, il s'agit du cas le plus consommant et durant lequel le message prend le plus de temps à être envoyé ce qui nous permet d'échantillonner plus facilement les consommations. Dans notre test l'émetteur envoie un message toutes les 5 secondes.

TXPOWER = 1

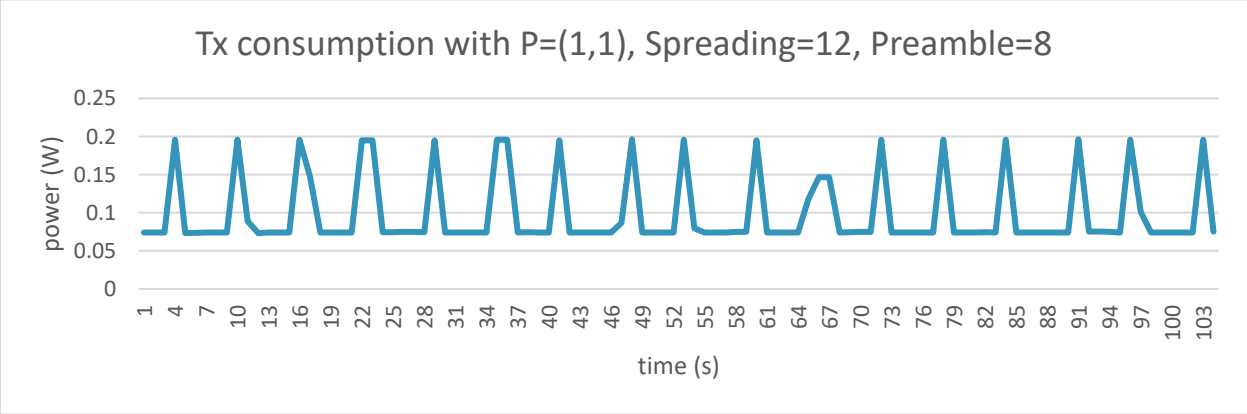

Figure 5- Consommation TxPower=1 et SF=12

Consommation de 200 mW à chaque envoi.

TXPOWER = 10

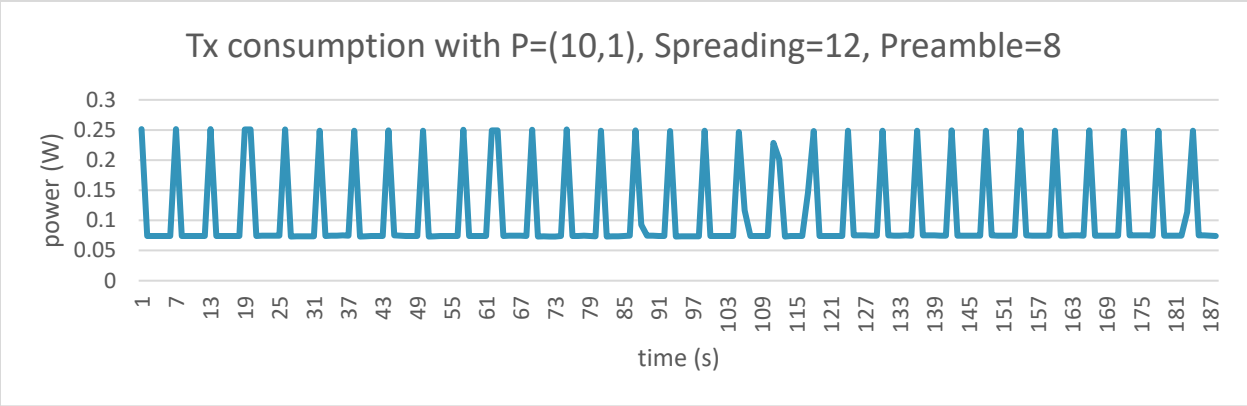

Figure 6- Consommation TxPower=10 et SF=12

Consommation de 250 mW à chaque envoi.

TXPOWER = 20

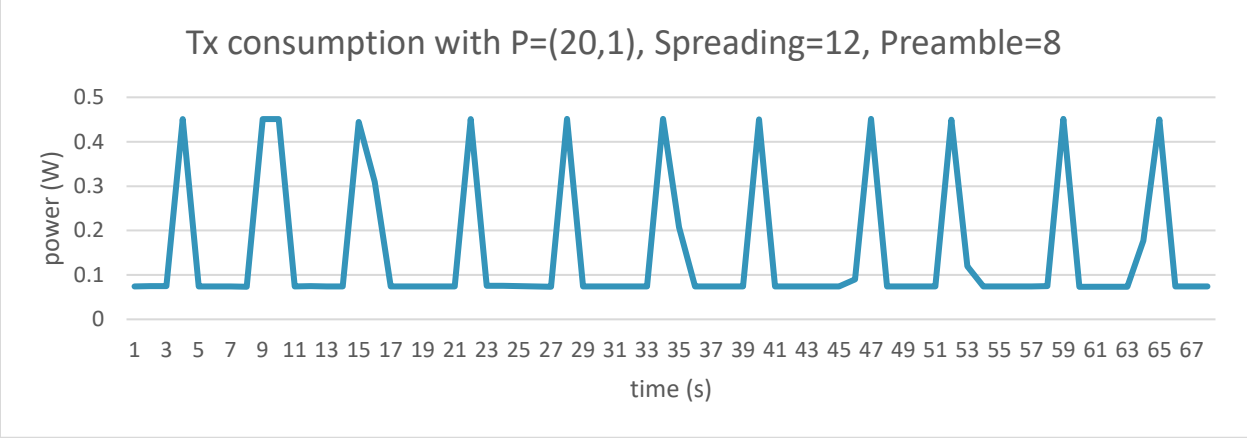

Figure 7- Consommation TxPower=20 et SF=12

Consommation de 450 mW à chaque envoi.

2.3 INFLUENCE DU SPREADING FACTOR SF

On mesure ensuite l'impact du SF sur la consommation. On configure la puce LoRa pour utiliser un TxPower de 20 et pour envoyer un message toutes les 5 secondes.

Etant donné que le module UM25C n'effectue qu'une mesure par seconde, les consommations de certains SF ne peuvent pas être correctement évaluées. En effet, pour des SF plus petits que 12 le temps « on air » est de moins d'une seconde. On se retrouve donc avec un problème de sous-échantillonnage. Pour pallier ce problème, on effectue des mesures d'au moins une minute pour examiner un grand nombre d'envois. Le pic de consommation maximal durant toute la mesure est pris comme référence pour la consommation d'un envoi.

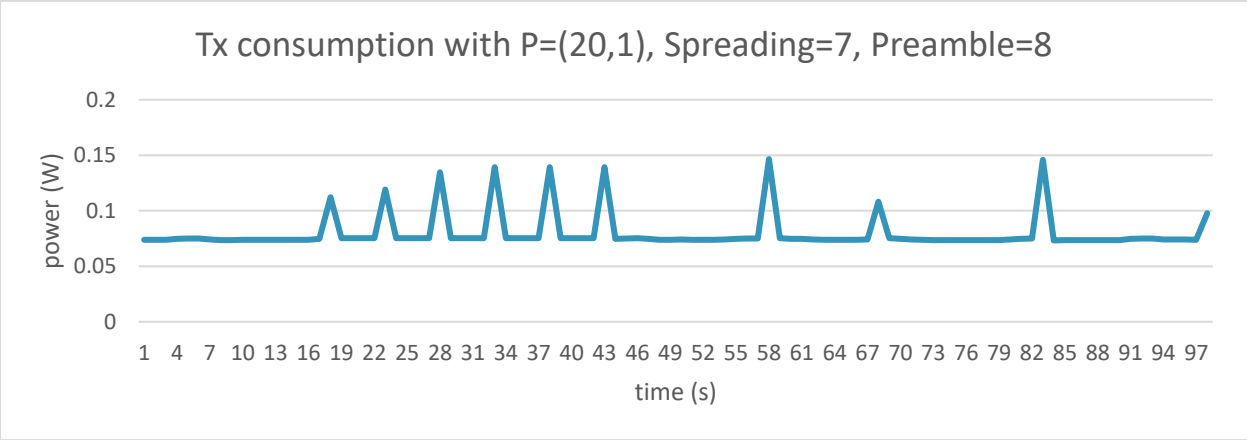

Figure 8- Consommation TxPower=20 et SF=7

Consommation de 150 mW à l'envoi.

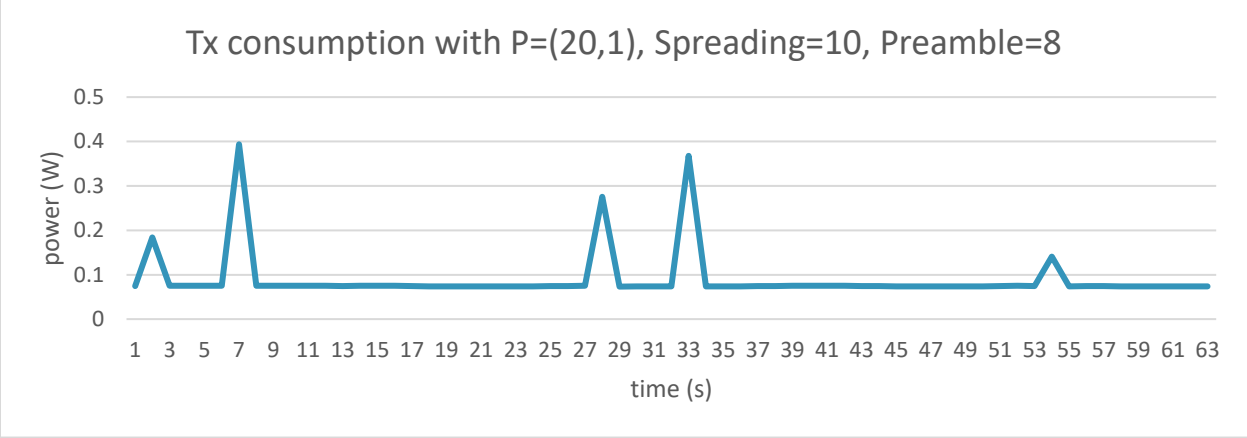

Figure 9- Consommation TxPower=20 et SF=10

Consommation de 400 mW à l'envoi.

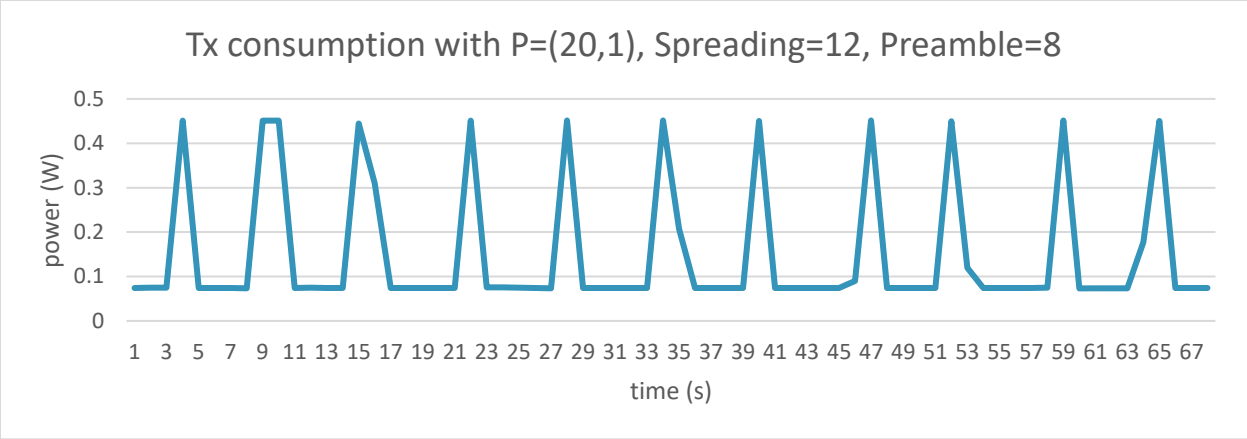

Figure 10- Consommation TxPower=20 et SF=12

Consommation de 450 mW à l'envoi.

2.4 MODES LOW POWER

Une partie de la consommation mesurée est due à la consommation statique de la carte. Lorsque le programme attend et ne fait rien, il est préférable de couper certaines fonctionnalités pour réduire cette consommation statique. Le microcontrôleur ATMega328p propose des modes « low power » qui lui permettent de réduire sa propre consommation. On étudie la consommation de chacun de ces modes pour déterminer leur impact sur la consommation totale.

Idle	PowerDown	PowerSave	Standby
20,4 mA	19,0 mA	19,4 mA	19,2 mA

Les modes basse consommation du microcontrôleur ne réduisent que très peu la consommation totale. Cela signifie que le microcontrôleur n'est pas la source principale de la consommation sur la plateforme. Implémenter l'utilisation des modes low power présente donc peu d'intérêt dans notre cas.

Une solution serait d'utiliser une carte plus customisée, avec très peu de modules superflus qui seraient plus facilement coupés.

SCENARIO APPLICATIF – GIROUETTE CONNECTEE

Nous imaginons un projet applicatif dans lequel l'utilisation du LoRA est pertinente.

La girouette d'un voiler est relié à un potentiomètre (ou un joystick), permettant de déduire l'angle du bateau par rapport au vent.

Notre système récupère cet angle, et s'il est trop éloigné du centre, log cette valeur pour un futur visionnage et une potentielle analyse des performances de la part de l'utilisateur.

En pratique, une notification doit être envoyé toute les minutes au moins tant que l'angle n'est pas rectifié, mais pour tester notre application ce temps sera réduit à 1 seconde.

Chaque trame LoRA contiendra la donnée d'angle. Il sera ensuite du ressort du programme récepteur de notifier l'utilisateur à chaque fois qu'une trame est reçue.

La distance entre les 2 modules LoRA ne dépassant pas les 20 mètres, le spreading factor et la puissance peuvent être mis au minimum.

LoRaSender

```
#include <SPI.h>
#include <LoRa.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>

#define LORA_KEY "QR_pkg-"

#define DEBUG

const uint16_t angle_delta = 200;

int JoyStick_X = A0; // Signal de l'axe X
int JoyStick_Y = A1; // Signal de l'axe Y
int Button = 5; // Bouton
int counter = 0;

uint16_t x, prev_x;
uint16_t y, prev_y;
uint16_t x_center;
int8_t button_pressed, prev_b;

void setup() {
#ifdef DEBUG
  Serial.begin(9600);
  while (!Serial);
  Serial.println("LoRa Sender");
#endif

  pinMode (JoyStick_X,INPUT);
  pinMode (JoyStick_Y,INPUT);
  pinMode (Button,INPUT);
  digitalWrite(Button,HIGH);

  if (!LoRa.begin(866E6)) {
#ifdef DEBUG
    Serial.println("Starting LoRa failed!");
#endif
    while (1);
  }
  LoRa.setTxPower(1,1);
  LoRa.setSpreadingFactor(7);
  LoRa.setPreambleLength(8);

  // x calibration
  delay(10);
  x_center =  x = analogRead(JoyStick_X);
}
```

Figure 11- Programme LoRaSender partie 1

```
void loop() {

  x = analogRead(JoyStick_X);
  y = analogRead(JoyStick_Y);
  button_pressed = digitalRead(Button);

#ifdef DEBUG
  if(abs(prev_x-x) > 1 || abs(prev_y-y) > 1 || abs(prev_b-button_pressed) > 1){
    Serial.print("X :"); Serial.print(x);
    Serial.print("    Y :"); Serial.print(y);
    Serial.print("    Button :");
    Serial.println(button_pressed);
  }
#endif
    // send packet
    if (angle_threshold_reached(x))
      loRa_send_onedata(x, y, button_pressed);
    prev_x=x; prev_y=y; prev_b=button_pressed;
    counter++;
  delay(1000);

}

bool angle_threshold_reached (uint16_t x){
  return abs(x-x_center) > angle_delta;
}

void loRa_send_onedata(uint16_t x, uint16_t y, int8_t b){
  LoRa.beginPacket();
    LoRa.print(LORA_KEY);
    LoRa.print(x);
    LoRa.print(";");
    LoRa.print(y);
    LoRa.print(";");
    LoRa.print(b);
    LoRa.println(";");
    LoRa.endPacket();
}
```

Figure 12- Programme LoRaSender partie 2

Comme vu ci-dessus sur le code, une trame n'est envoyée par LoRA que si l'angle est considéré comme trop éloigné. Cela permet de réduire la consommation du module.

Dans ce code, deux angles sont envoyés car nous utilisons un joystick, mais un seul est nécessaire. Le bouton poussoir est utilisé pour stopper l'application côté client.

LoRaReceiver

```
#include <SPI.h>
#include <LoRa.h>

#define LORA_KEY_SIZE 7
#define LORA_KEY "QR_pkg-"

void setup() {
  Serial.begin(9600);
  while (!Serial);

  //Serial.println("LoRa Receiver");

  if (!LoRa.begin(866E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  //  LoRa.setTxPower(20,1);
  LoRa.setSpreadingFactor(7);
  LoRa.setPreambleLength(8);
}

void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {

    String read_data;
    while (LoRa.available()) {
      read_data +=(char)LoRa.read();
      //Serial.print((char)LoRa.read());
    }
    if(read_data.substring(0,LORA_KEY_SIZE) == LORA_KEY){
      Serial.print(read_data.substring(LORA_KEY_SIZE));
    }
  }
}
```

Figure 13- Programme LoRa Receiver

Le récepteur ne fait que forwarder les messages reçus à l'application Python en faisant le tri parmi d'autres potentielles applications LoRA (grâce à LORA_KEY).

```python
import serial as s
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt



def initSerial(baudrate, port):
    ser = s.Serial()
    ser.baudrate = baudrate
    ser.port = port
    ser.timeout = 10
    return ser


NB_FRAME = 1000
frame = []

ser = initSerial(9600, "COM18")
ser.close()
ser.open()
t = np.array(dt.datetime.now())

file = open("log/log_data.csv", "w")

keepalive = '1';

while(keepalive != '0'):
    if ser.inWaiting():
         date = dt.datetime.now()
         frame_str = str(ser.readline())[2:]
         t = np.append(t, date)
         buff = frame_str.split(';')
         frame.append([buff[0], buff[1]])
         keepalive = buff[2]
         csv_string = str(date.time()) + "," + str(buff[0]) + '\n'
         print(csv_string)
         file.write(csv_string)

file.close()
ser.close()
plt.show()
```

Figure 14-Application Python

L'application Python permet de logger toute les valeurs reçus en fonction du temps au format csv. On peut imaginer par la suite une application de formatage qui permettrait à l'utilisateur de visionner les statistiques de sa sortie en mer.

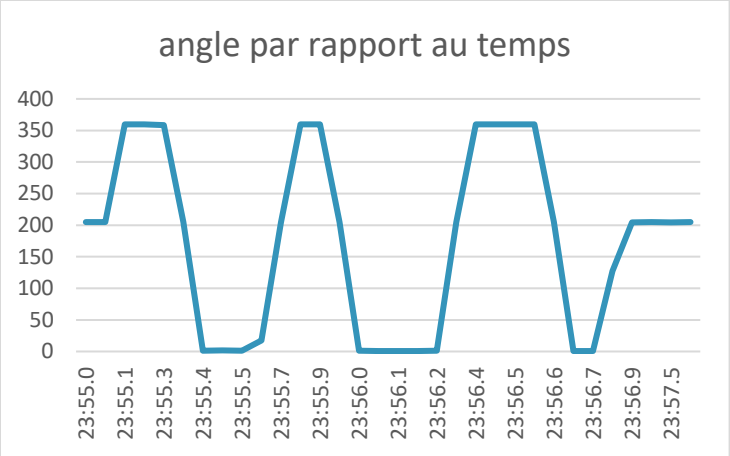

Figure 15- Exemple de résultat de l'application

2.5 ANALYSE DE LA CONSOMATION

On va calculer une approximation de l'autonomie de notre système, basé sur la consommation mesurée de la carte et l'utilisation d'une batterie de 1200mAh -> 3960mJ (tension de sortie de 3.3V).

On relève une consommation pic de 40mA et une consommation normale de 22.4mA, sachant la tension d'alimentation à 3.3V.

Si l'on envoie aucune donnée, l'autonomie va être de :

$3.3 * 22.4 = 73.92mW$ -> $\frac{3960}{73.92} = 53.57H$ -> 2 jours 5 heures et 34 minutes.

Si l'on envoie des données sans arrêt, l'autonomie va être de :

$3.3 * 40 = 132mW$ -> $\frac{3960}{132} = 30H$ -> 1 jour 6 heures.

Donc dans le pire cas notre système peut tenir une journée en mer, mais on peut partir du principe que l'utilisateur va devoir changer la batterie tous les jours.

Voyons maintenant la fréquence de messages maximum nécessaire pour tenir 2 sorties en mer de 10 heures espacées de 24 heures (scénario arbitraire):

- 24H sans envoi de données -> $1200 - 22.4 * 24 = 662.4mAh$ restant sur la batterie.
- 20H en fonctionnement normal -> $I_{idle} * (20H - T_{run}) + I_{run} * T_{run} = 662.4mAh \rightarrow T_{run} = 12H$ soit 6H par sorties.

Dans ce scénario le système peut envoyer des données la moitié du temps et tenir suffisamment longtemps pour 2 sorties en mer.

3 CONCLUSION

Le LoRA est un protocole de communication idéal pour les applications low power. Nous avons pu voir avec notre girouette connectée que cette technologie fonctionne bien pour les courts messages envoyés par pulse.

Notre application est utilisable tel quel, mais pourrait être améliorée en développant une carte customisée qui s'affranchirait de toute fonctionnalité superflue (comme le module de conversion de tension). Une augmentation de la capacité de la batterie peut aussi être envisagée, pour éviter à l'utilisateur de devoir monter régulièrement en haut du mât pour changer la batterie.

On pourrait aussi imaginer un mode StandBy, qui correspondrait à l'amarrage au port, et dans lequel aucune donnée n'est envoyé, peu importe l'angle du mât.