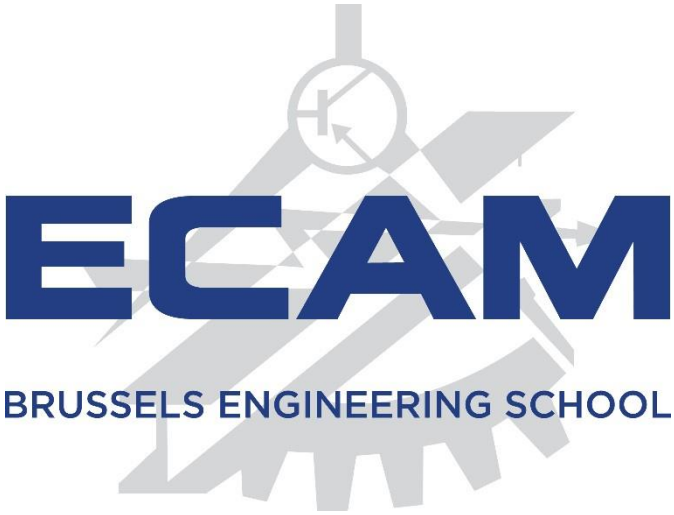
Rapport d'IOT



Zakaria Boukarsa (17338)
Bastien Van Den Neste (16327)
Abdoulaye koolwx (14084)
Rekik Heni (16281)

Professeur

Philippe Dekimpe

Table des matières

I. Description du projet

Vu les circonstances actuelles, nous avons réalisé “Festi.People” qui servira au ralentissement de la propagation du virus dans des futurs grands évènements.

Festi.People adopte une architecture IOT qui sera déployé dans un emplacement géographique bien déterminé, l'utilité de notre projet est de quantifier le flux de personnes en entrée et en sortie afin de contrôler et monitorer le lieu.

Festi.People va créer une interconnexion entre notre lieu et internet via un réseau de capteurs.

Cette interconnexion se caractérise par l’ensemble de fonctionnalités suivantes :

- La première fonctionnalité est déterminée le sens du mouvement pour faire la distinction entre une entrée et une sortie.
- La deuxième fonctionnalité s’agit de transférer les données vers un serveur web de façon unidirectionnelle, confidentielle et distante.
- La dernière fonctionnalité sert à traiter les données dans une interface adaptée pour afficher les données.

Toutes les fonctionnalités seront détaillées dans la suite de ce rapport. Les tâches qui seront exécutées par notre projet doivent être assurer pendant une longue durée de temps pour éviter toutes sortes de pertes d’informations.

II. Description de l’architecture mise en place

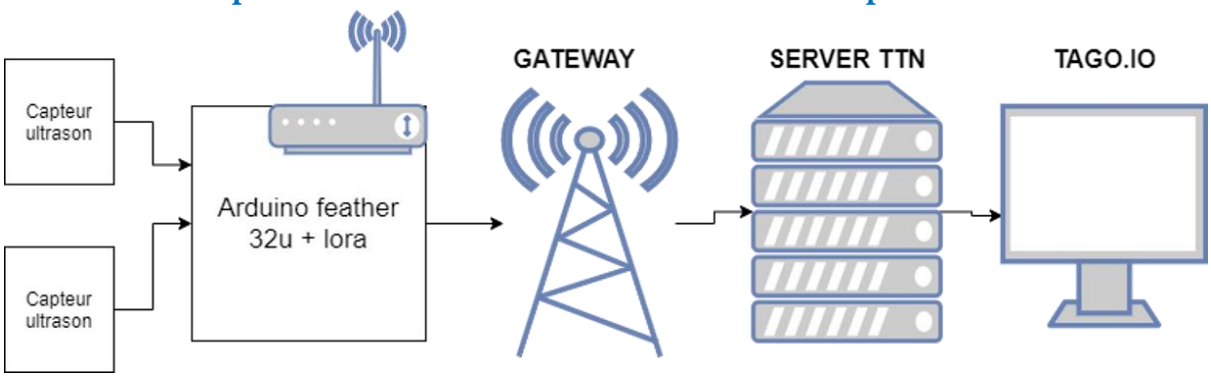


Figure 1

La figure 1 illustre bien l’architecture générale de notre projet.

Il y aura donc deux capteurs ultrasons qui détecterons les passages aux portes du festival et les signaleront à l’Arduino.

Celui-ci traitera les informations qu’il recevra des capteurs pour savoir si un passage correspond à une entrée ou une sortie en fonction du capteur qui détectera le passage en premier.

L'Arduino fera donc le comptage des entrées et des sorties pendant cinq minutes avant d'envoyer ces données à la Gateway grâce la technologie LoRa et ensuite réinitialiser ces compteurs.

La Gateway elle va servir à faire transférer les données d'entrée et sortie au server *The Things Network* qui lui va décoder le payload reçu par la Gateway pour avoir les entrées et les sorties en décimal pour ensuite envoyer ces informations au server applicatif *TAGO.IO* qui lui va afficher ces informations.

III. Hardware

1. Liste des composants

- 2 Capteurs ultrasonic - HC-SR04 (Generic)
- 2 Breadboard
- 1 Arduino feather 32u4 Lora
- Fils de connexion
- 1 Cable USB-A vers Micro-USB

2. Description du montage

Nous utiliserons comme alimentation la sortie USB de l'*Arduino feather 32u* qui renvoie 5V pour les capteurs ultra son. Cependant, nous avons utilisé des résistances afin de protéger notre *Feather* en les rajoutant en série au *echo*.

Nous utilisons les broches pins PWM de la *Feather* pour les connecter aux broches *echo* et *trigger* :

- Capteur 1 (mesurant l'entrée d'une personne) : pin 5 pour *echo* et 11 pour *trigger*
- Capteur 2 (mesurant la sortie d'une personne) : pin 9 pour *echo* et 12 pour *trigger*

De plus, nous avons utilisé 2 breadboards afin d'avoir une distance assez éloignée entre les capteurs afin de pouvoir mesurer correctement la distance. En effet, si cette distance est trop faible il se pourrait qu'on ne puisse pas détecter le sens de passage.

IV. Software

1. Installation et librairies

a. Capteurs + Arduino

Nous avons utilisé l'IDE arduino afin de programmer nos capteurs. La principale difficulté est de coder afin de déterminer le sens de passage. Nous expliquerons plus en détail le code dans la section suivante.

b. The Thing Network

On a utilisé le compte the thing network de l'ecam pour ensuite créer notre application ***festipeuple*** dans lequel on a enregistré un device nommé ***festi_peuple***.

La procédure pour créer une application et y enregistrer un device est mentionné sur le site de *the thing network* https://www.thethingsnetwork.org/docs/devices/node/quick-start.html.

On a configuré ce device pour activer la méthode ***ABP***. On a suivi les instructions du site de *The Thing Network* https://www.thethingsnetwork.org/docs/devices/registration.html#personalize-device-for-abp.

Sachant que le payload reçu sur le server TTN est en bytes, on a dû décoder ce payload pour avoir nos données en décimal pour ensuite envoyer ces données à notre server applicatif *TAGO.IO*. Pour cela nous avons mis le script JavaScript de *l'annexe 1* dans notre application.

c. LoRa + Arduino

Comme dit plus haut on utilise *l'Arduino Feather 32u + LoRa* qui intègre donc une puce LoRa et une antenne pour communiquer via la technologie LoRa.

La première chose qu'on a fait a été de configurer l'IDE Arduino comme sur le site https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-module/setup.

Pour utiliser LoRa nous avons utilisé la bibliothèque *TinyLora*. Etant en Europe on a d'abord sélectionné la configuration Européenne de notre librairie *TinyLora* en modifiant son fichier TinyLora.cpp. On a commenté le code *#define US902* à la ligne 72 de ce fichier et décommenté le code *#define EU863* de la ligne 73.

2. Explication des scripts

a. Lora + Arduino

```
// Visit your thethingsnetwork.org device console
// to create an account, or if you need your session keys.

// Network Session Key (MSB)
uint8_t NwkSkey[16] = { 0x08, 0x9E, 0x30, 0x2A, 0xAF, 0x1F, 0xC4, 0xA0, 0xD8, 0x09, 0x4B, 0x23, 0x73, 0x00, 0xD6, 0x00 };

// Application Session Key (MSB)
uint8_t AppSkey[16] = { 0x5C, 0x47, 0x7F, 0x00, 0xE7, 0xF6, 0xF4, 0x74, 0xD7, 0xE4, 0xE9, 0x8C, 0x54, 0x00, 0x04, 0x00 };

// Device Address (MSB)0
uint8_t DevAddr[4] = { 0x26, 0x01, 0x37, 0x65 };
// uint8_t DevAddr[4] = { 0x26, 0x01, 0x1A, 0x72 };
```

Figure 2

Concernant le script du code Arduino pour la partie LoRa, on a dû d'abord importer la bibliothèque TinyLora avant de renseigner les informations de notre application et de notre device comme sur *la figure 2*.

```
// Initialize LoRa
Serial.print("Starting LoRa...");
// define multi-channel sending
lora.setChannel(MULTI);
// set datarate
lora.setDatarate(SF7BW125);
while(!lora.begin())
{
  Serial.println("Failed");
  Serial.println("Check your radio");
  //while(true);
  delay(5000);
}
```

Figure 3

A la *figure 3* on configure notre bibliothèque Tiny pour avoir une portée d'un +-1 km, c'est-à-dire un Spreading Factor de 7 et une bande passante de 125.

```
void sendata(int16_t in, int16_t out){
  // encode int as bytes
  //byte payload[2];
  Serial.println(in);
  Serial.println(out);
  loraData[0] = highByte(in);  // left byte, lower value
  loraData[1] = lowByte(in);

  loraData[2] = highByte(out);
  loraData[3] = lowByte(out);

  Serial.println("Sending LoRa Data...");
  lora.sendData(loraData, sizeof(loraData), lora.frameCounter);
  Serial.print("Frame Counter: ");Serial.println(lora.frameCounter);
  lora.frameCounter++;
}
```

Figure 4

La *figure 4* montre la fonction *sendata*. Cette fonction prend en paramètre 2 variables, une représente la variable des entrées et l'autre la variable des sorties et permet d'envoyer ces variables à la Gateway.

On met donc dans la variable globale de 4 bytes ***loraData*** les entrées et les sorties qu'on envoie ensuite à la Gateway via la fonction *sendData* de la bibliothèque *Tiny*.

b. Gestion des capteurs

peopleCounter §

```
int currentPeopleIn= 0;
int currentPeopleOut= 0;


int sensor1[] = {5,11};
int sensor2[] = {9,10};
int sensor1Initial=0;
int sensor2Initial=0;

String directionPassage = "

int timeoutCounter = 0 ;
```

Figure 5

Les 2 variables *currentPepopleIn* et *currentPeopleOut* sont nos variables qui s'incrémenteront lors du passage d'une personne. Ce sont donc nos variables que nous enverrons au server TTN via la Gateway. Ensuite, nous initialisons les 2 capteurs ultrasons en mettant les pins adéquates de notre Arduino Feather 32u. Nous avons aussi une variable *directionPassage* qui

est de type *String*, et permet de stocker le sens afin d'incrémenter une entrée ou une sortie. Pour finir, *timeCounter* permet de réinitialiser la direction si elle a des délais d'attentes trop long comme par exemple une personne qui reste devant un des capteurs.

```
void setup() {
  //Setup code
  Serial.begin(9600);

  delay(100);
  sensor1Initial = measureDistance(sensor1);
  sensor2Initial = measureDistance(sensor2);
}
```

Figure 6

Le setup est constitué simplement d'un serial afin d'afficher nos valeurs sur notre pc afin d'observer les états de nos différentes variables. De plus, nous initialisons 2 variables *sensor1Initial* et *sensor2Initial* à l'aide d'une fonction. Cette assignation nous permet d'obtenir la distance en cm et d'avoir la distance initiale entre les 2 obstacles qui caractérisent l'entrée et la sortie (par exemple une porte, un portique).

```
void loop() {
  //Read ultrasonic sensors
  int sensor1Val = measureDistance(sensor1);
  int sensor2Val = measureDistance(sensor2);

  //Process the data
  if(sensor1Val < sensor1Initial - 20 && directionPassage.charAt(0) != '1'){
    directionPassage += "1";
    delay(100);
  }else if(sensor2Val < sensor2Initial - 20 && directionPassage.charAt(0) != '2'){
    directionPassage += "2";
    delay(100);
  }

  if(directionPassage.equals("12")){
    currentPeopleIn++;
    directionPassage="";
  }else if(directionPassage.equals("21")){
    currentPeopleOut++;
    directionPassage="";
  }

  //Resets the direction if it is invalid or timeouts
  if(directionPassage.length() > 2 || directionPassage.equals("11") || directionPassage.equals("22") || timeoutCounter > 200){
    directionPassage="";
  }
  if(directionPassage.length() == 1){ //
    timeoutCounter++;
  }else{
    timeoutCounter=0;
  }
  //Print values to serial
  Serial.println(currentPeopleIn);
  Serial.println(currentPeopleOut);
}
```

Figure 7

Dans un premier, nous voyons que nous définissons 2 autres variables par la même fonction que dans le setup. Ici, ça permet de détecter une personne. En effet, si la différence entre la valeur initialisée et 20 cm qui correspond à l'envergure d'un individu est supérieure à la valeur de la distance mesuré dans la loop alors il s'agit d'une personne qui est passée devant un des capteurs. On pourra ajouter dans la séquence le chiffre 1 ou 2 en fonction du capteur qui a détecté cette variation.

Ensuite, nous posons la condition sur le String *directionPassage* afin de savoir de quelle séquence il s'agit. Si cette variable vaut '12', alors il s'agit d'une personne qui entre. On a donc que si la séquence vaut '21', elle sort.

Pour finir, nous avons plusieurs conditions afin de reseter la séquence si plusieurs problèmes survenaient et nous affichons les 2 variables *currentPeopleIn* et *currentPeopleOut* dans le serial.

c. Détails TAGO.io

Pourquoi tago.io ?

Notre système est par définition unidirectionnel, il est inutile pour nous d'envoyer des informations vers l'Arduino, tout ce qui nous importe c'est de récupérer des informations, les traiter et les afficher.

Tago.io permet est basé sur un système de Dashboard sur lequel on vient ajouter des widgets personnalisés qui nous permettent d'afficher les différentes variables selon le format que l'on choisit (compteur, graphique, jauge, ...).

Tago.io dispose aussi d'un système de scripts Node.js qui permettent d'effectuer des opérations sur les valeurs envoyées par les capteurs et de stocker certaines variables globales sur l'application elle-même. Et enfin pour compléter le tout, le site nous permet aussi de créer des alertes qui pourront entre autres activer des scripts Node.js automatiquement.

Dans notre implémentation, Le Dashboard contiendra trois compteurs qui afficheront : le dernier lot de personnes qui sont rentrés, le dernier lot de personnes qui sont sorties, le nombre total de personnes à l'intérieur.

Deux scripts node.js seront créés, un qui récupèrera les deux valeurs envoyées par les capteurs (**CHECKTOTAL**) :

- Les entrées (*entrence*) et les sorties (*out*) et qui incrémentera ou décrémentera la variable globale *totPpl* en fonction de *entrance* et *out.*
- Le second script (**RESETTOTAL**) resettera la variable *totPpl* à zéro.

Deux alertes seront connectées aux deux scripts python qu'ils lanceront lorsque les conditions suivantes seront remplies :

- Pour **CHECKTOTAL** à chaque fois qu'une des Devices connectées à notre application enverra une donnée (toutes les 5 minutes environ) afin de mettre à jour le nombre total de personnes à l'intérieur
- Pour **RESETTOTAL** tous les jours à la fin de l'évènement (3 heures du matin par exemple) pour remettre le compteur à zéro pour le jour suivant.

3. Mise en place de l'application

Tago se base sur un système de Bucket dans lequel on peut mettre un certain nombre de Devices (dans notre prototype nous n'avons qu'une seule Device, mais en application on aura tout un set de device), un Bucket contiendra toutes les informations sur les Devices dont les données envoyées et permettera de les présenter sur un Dashboard.

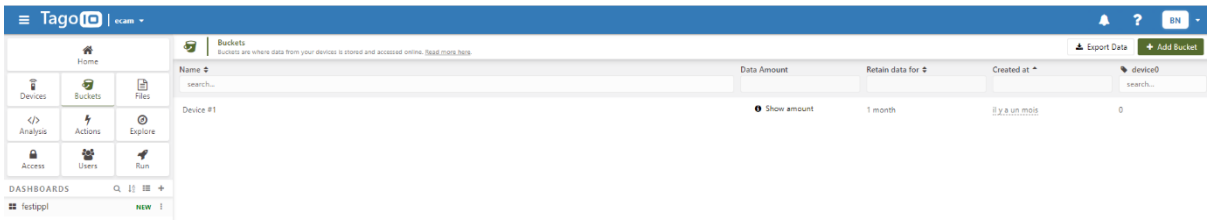


Figure 8

Pour relier une Device à un Bucket il faut récupérer le Device EUI de l'arduino

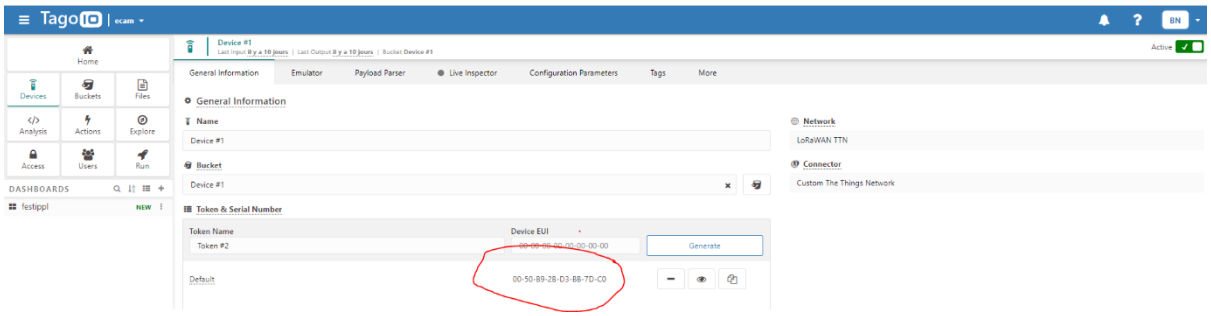


Figure 9

Une fois que la Device est connectée et qu'elle a envoyé un premier message on peut récupérer toutes les données qui ont été traitées et envoyées sous format Json depuis TTN

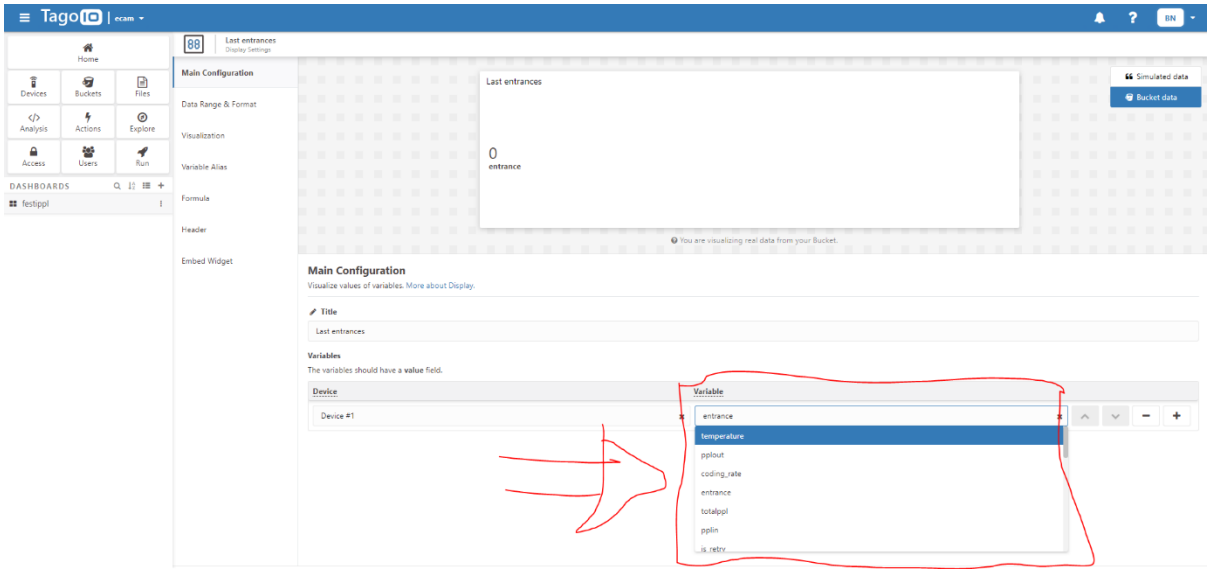


Figure 10

Ainsi qu'un certain nombre de métadonnées sur les messages et l'appareil (nous ne les utiliseront pas).

4. Scripts node.js

CHECKTOTAL

Dans les scripts node.js on récupère les trois valeurs présentent dans le Bucket de l'application :

```
28  // create the filter options to get the data from TagoIO
29  const filter = {
30    variable: "entrance",
31    query: "last_item",
32  };
33
34  const filter2 = {
35    variable: "totPpl15_12",
36    query: "last_item",
37  };
38
39  const filter3 = {
40    variable: "out",
41    query: "last_item",
42  };
43
44  const resultArray = await device.getData(filter).catch(() => null);
45  const resultArray2 = await device.getData(filter2).catch(() => null);
46  const resultArray3 = await device.getData(filter3).catch(() => null);
```

Figure 11

Entrance et *out* sont les dernières valeurs d'entrées et sorties envoyées par l'Arduino.

TotPpl15_12 correspond au total actuel de personnes à l'intérieur.

Une fois que ces valeurs sont récupérées on modifie *totPpl* en fonction de *out* et *entrance*.

```
53  // query:last_item always returns only one value
54  const pplinValue = resultArray[0].value;
55
56  const totValue = resultArray2[0].value;
57
58  const pploutValue = resultArray3[0].value;
59
60  const obj_to_save = {
61    variable: "totPpl15_12",
62    value:totValue + (pplinValue-pploutValue),
63  };
```

Figure 12

Et enfin, on met à jour la valeur totPpl.

```
70  try {
71    await device.sendData(obj_to_save);
72    context.log(
73      `Successfully Inserted, entrance : ${pplinValue} , out : ${pploutValue}`
74      );
75    context.log(
76      `old total : ${totValue}`
77      );
78  } catch (error) {
79    context.log("Error when inserting:", error);
80  }
```

Figure 13

RESETTOTAL

Le script *resetTotal* est globalement le même, on vient récupérer uniquement *totPpl* et on le met à jour à 0.

```
33  const obj_to_save = {
34    variable: "totppl15_12",
35    value: 0,
36  };
```

Figure 14

5. Dashboard

Chaque widget peut être relié à une des valeurs du Bucket

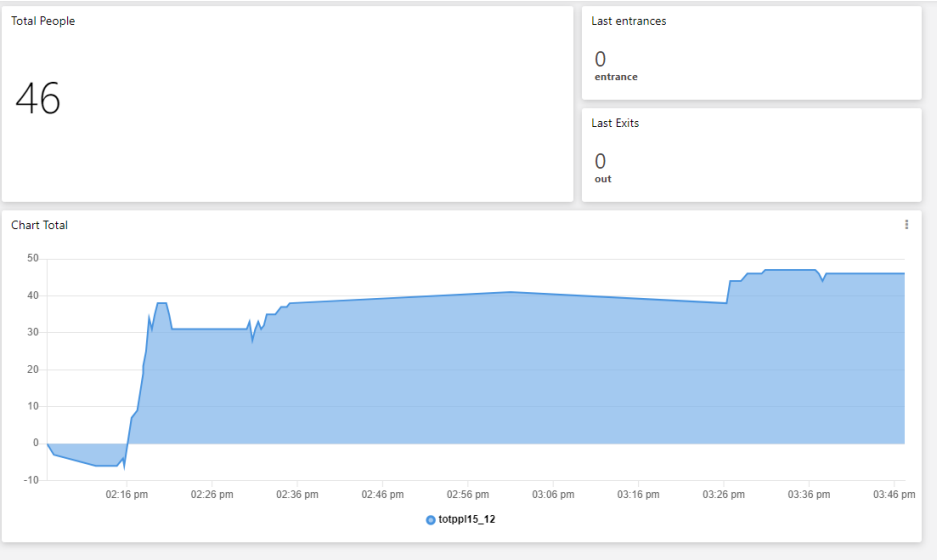


Figure 15

On a simplement trois compteurs qui affichent *totPpl*, *entrence* et *out*. Et un graphique qui suit l'évolution de *totPpl*, voici sa configuration :

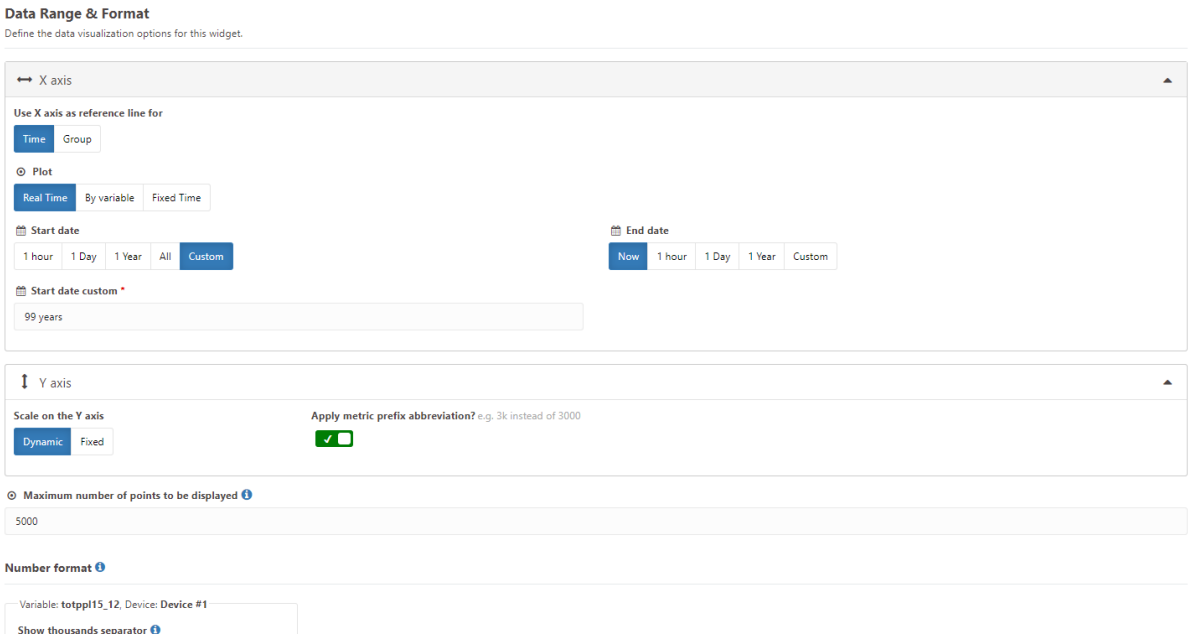


Figure 16

6. Alertes

Voici la configuration de l'alerte qui activera *checkTotal*.

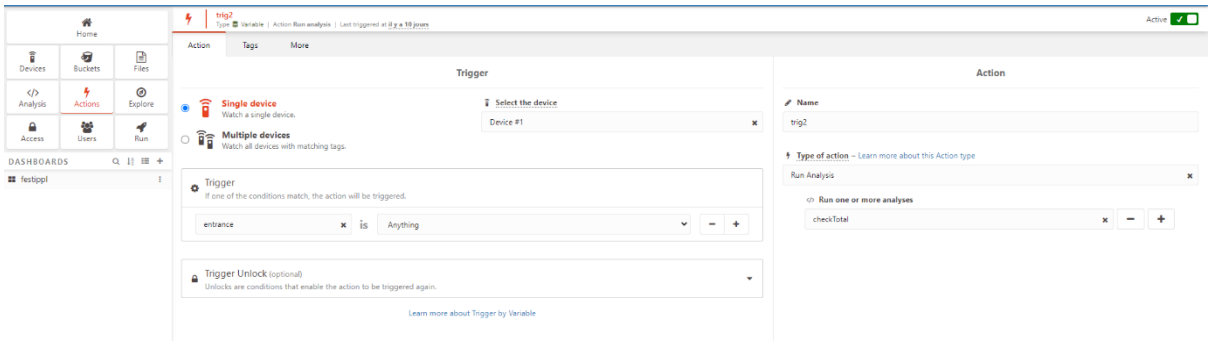


Figure 17

Il est activé dès que la valeur *entrance* du Bucket est changée (donc à chaque envoi de données)

V. Optimisation portée et consommation

Pour optimiser la consommation, nous nous sommes principalement intéressés à la mise en veille de notre Arduino Feather 32u. En effet, nous utilisons une payload de **(à préciser)** qui est déjà un bon compromis. De plus, nous ne sommes pas inquiets à propos de la portée car en général la Gateway sera disponible dans le festival et avoir un rayon de 1km soit un Spreading Factor de 7 est acceptable.

1. Essais réalisés

Nous voulions dans un premier temps savoir ce que consommait notre Feather à lui seul. Pour ce faire, nous avons dénudé une partie du câble afin d'y connecter le multimètre est d'observer l'envoie du message à la Gateway et sa consommation.

Ensuite, nous y avons rajouté les capteurs ultrasons. Nous avons procédé au même montage que précédemment. Nous voulons apercevoir ce que les capteurs demandent en énergie.

Une fois ces 2 expériences faites, nous avons essayé de réaliser des mesures lorsque la Feather pouvait s'endormir au bout de 10 secondes d'inactivité afin d'observer notamment la consommation de la carte en veille. Pour ce faire, nous utilisons un capteur infrarouge et l'ajout de la fonctionnalité mode *sleep* qu'on a programmé dans l'IDE de l'Arduino en utilisant la bibliothèque ***avr/sleep.h***.

VI. Résultats obtenus et interprétations

Les résultats obtenus pour la 1ère expérience :

Figure 18

Nous voyons sur la figure 18 que la consommation moyenne de l'Arduino est de 12mA et que l'envoi de la payload à la Gateway est de 6 mA. Sachant que dans la pratique, le message se fera plus rarement par exemple toutes les 30minutes, on a essayé d'optimiser en endormant l'Arduino car sa consommation était importante et inutile lorsqu'aucune personne ne passait.

Les résultats obtenus pour la 2eme expérience :

Figure 19

Ici (Figure 19), nous observons que les capteurs demandent plus de 5mV pour fonctionner et la consommation moyenne est de 17.5mV. Nous pouvons ainsi déjà faire une approximation de la durée d'une batterie de 3200mAh.

$$\frac{3200}{17.5} \cong 187 heures \cong 7.8 jours$$

Notre dispositif pourrait tenir environs 8 jours sans s'arrêter ce qui est déjà suffisant pour un festival.

Les résultats obtenus pour la 3eme expérience :

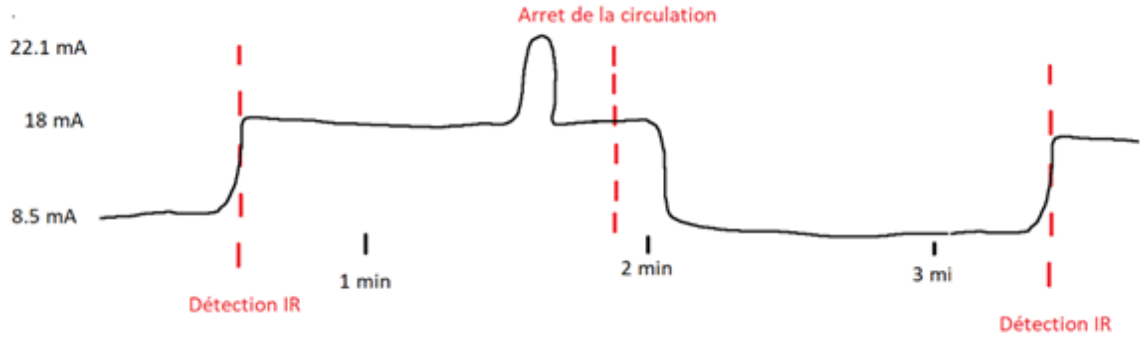


Figure 20

À la suite de cette expérience, nous remarquons déjà une grande différence dans la consommation. En mode *sleep*, elle est de 8.5mA. On voit aussi que le capteur infrarouge détecte bien le passage d'une personne et allume la Feather afin d'effectuer les mesures et d'envoyer le message.

Comme précédemment, nous pouvons effectuer une approximation de la durée de vie d'une batterie de 3200mAh. Cependant, la consommation est dépendante de la fréquentation. Prenons 2 cas possibles :

- Consommation moyenne avec peu d'affluence (Arduino endormi 50% du temps) 13.25 mAh

$$\frac{3200}{13.25} \cong 241.5heures \cong 10jours$$

- Consommation moyenne avec beaucoup d'affluence (Arduino endormi 10% du temps) 16.2 mAh

$$\frac{3200}{16.2} \cong 197.53heures \cong 8.23jours$$

Nous observons que la durée a augmenté lors de l'ajout de cette fonctionnalité. De plus, dans ces calculs nous n'avons pas pris en compte les heures creuses où aucune personne ne passera comme par exemple tard dans la soirée jusqu'à l'aube ce qui augmentera considérablement la durée de vie.

VII. Conclusion et pistes d'amélioration

Nous avions comme objectif de créer un système capable de monitorer les entrées sorties dans un lieu donné avec une bonne fiabilité, une grande autonomie et la possibilité de monitorer plusieurs entrées simultanément.

Festi.People remplit tous ces critères sans problèmes, notre monitoring est fiable : TAGO.IO, TTN et l'Arduino Feather 32u n'ont pratiquement pas de pertes d'informations.

Notre système est capable de tenir sans recharges plusieurs jours avec une petite batterie de 3200mh et le système de buckets de Tago.io nous permet de synchroniser les différentes devices à l'application avec simplicité.

Comme piste d'amélioration au niveau de l'Arduino, nous pourrions ajouter un pin permettant de récupérer le niveau de batterie et de l'envoyer vers le réseau LoRa.

Au niveau de l'application on pourrait avoir un système d'alertes qui enverrait une notification au gérant du festival (système proposé par Tago.io) lorsque le niveau de batterie d'un capteur devient trop faible ou lorsque le nombre de personnes dans le festival est supérieur à un *threshold* donné.

VIII. Annexes

1. Script decoder payload TTN

```
function Decoder(bytes, port) {
    // Decode an uplink message from a buffer
    var e=(bytes[0] << 8 | bytes[1]);
    var o=(bytes[2] << 8 | bytes[3]);

    return{
      entrance: parseInt(e),
      out: parseInt(o)
    };
}
```

2. Code Arduino

```
// TinyLoRa DHT22 - ABP TTN Packet Sender (Multi-Channel)
// Tutorial Link: https://learn.adafruit.com/the-things-network-for-
feather/using-a-feather-32u4
//
// Adafruit invests time and resources providing this open source code.
// Please support Adafruit and open source hardware by purchasing
// products from Adafruit!
//
// Copyright 2015, 2016 Ideetron B.V.
//
// Modified by Brent Rubell for Adafruit Industries, 2018
/************************ Configuration ******************************/
#include <TinyLoRa.h>
#include <SPI.h>
//#include "Adafruit_SleepyDog.h"

// Visit your thethingsnetwork.org device console
// to create an account, or if you need your session keys.

// Network Session Key (MSB)
uint8_t NwkSkey[16] = { 0x08, 0x9E, 0x30, 0x2A, 0xAF, 0x1F, 0xC4, 0xA0, 0xD8,
0x09, 0x4B, 0x23, 0x73, 0x00, 0xD6, 0x00 };

// Application Session Key (MSB)
uint8_t AppSkey[16] = { 0x5C, 0x47, 0x7F, 0x00, 0xE7, 0xF6, 0xF4, 0x74, 0xD7,
0xE4, 0xE9, 0x8C, 0x54, 0x00, 0x04, 0x00 };

// Device Address (MSB)0
```

```
uint8_t DevAddr[4] = { 0x26, 0x01, 0x37, 0x65 };
// uint8_t DevAddr[4] = { 0x26, 0x01, 0x1A, 0x72 };

/************************ Example Begins Here ******************************
*****/
// Data Packet to  to TTN
unsigned char loraData[4];

// How many times data transfer should occur, in seconds
const unsigned int sendInterval = 25;

// Pinout for Adafruit Feather 32u4 LoRa
TinyLoRa lora = TinyLoRa(7, 8, 4);

// Pinout for Adafruit Feather M0 LoRa
//TinyLoRa lora = TinyLoRa(3, 8, 4);

unsigned long start = millis();
int16_t in = 150;
int16_t out = 250;

/****************** Counter Variables ****************************/
int currentPeopleIn= 0;
int currentPeopleOut= 0;


int sensor1[] = {5,11}; //{echo, trigger}
int sensor2[] = {9,10};
int sensor1Initial=0;
int sensor2Initial=0;
int jeu;

String sequence = "";

int timeoutCounter = 0 ;

void setup()
{
  delay(1000);
  sensor1Initial = measureDistance(sensor1);
  jeu = measureDistance(sensor2);
  sensor2Initial = measureDistance(sensor2);
  Serial.begin(9600);

  // else doesn't start without serial monitor
  int waitcnt = 0;
  while(!Serial && (waitcnt++ < 10))  // wait (only so long) for serial port t
o connect.
  {
```

```
  delay(100);
  digitalWrite(LED_BUILTIN, waitcnt & 1);
  }

  // Initialize pin LED_BUILTIN as an output
  pinMode(LED_BUILTIN, OUTPUT);

  // Initialize LoRa
  Serial.print("Starting LoRa...");
  // define multi-channel sending
  lora.setChannel(MULTI);
  // set datarate
  lora.setDatarate(SF7BW125);
  while(!lora.begin())
  {
    Serial.println("Failed");
    Serial.println("Check your radio");
    //while(true);
    delay(5000);
  }
  Serial.println("OK radio success");
}

void loop()
{
  //Read ultrasonic sensors
  int sensor1Val = measureDistance(sensor1);
  int sensor2Val = measureDistance(sensor2);

  //Process the data
  if(sensor1Val < sensor1Initial - 20 && sequence.charAt(0) != '1'){
    sequence += "1";
    delay(550);
  }else if(sensor2Val < sensor2Initial - 20 && sequence.charAt(0) != '2'){
    sequence += "2";
    delay(550);
  }

  if(sequence.equals("12")){
    currentPeopleIn++;
    sequence="";
    delay(550);
  }else if(sequence.equals("21")){
    currentPeopleOut++;
    sequence="";
    delay(550);
  }
```

```
  //Resets the sequence if it is invalid or timeouts
  if(sequence.length() > 2 || sequence.equals("11") || sequence.equals("22") |
| timeoutCounter > 200){
    sequence="";
  }

  if(sequence.length() == 1){ //
    timeoutCounter++;
  }else{
    timeoutCounter=0;
  }

  //Print values to serial
  Serial.print("Seq: ");
  Serial.print(sequence);
  Serial.print(" S1: ");
  Serial.print(sensor1Val);
  Serial.print(" S2: ");
  Serial.println(sensor2Val);

  Serial.print(" people: ");
  Serial.println(currentPeopleIn);
  Serial.println(currentPeopleOut);

  // sending data
  if(millis() - start > sendInterval * 1000){
    sendata(currentPeopleIn, currentPeopleOut);
    currentPeopleIn = 0;
    currentPeopleOut = 0;
    start = millis();
  }

  // blink LED to indicate packet sent
  digitalWrite(LED_BUILTIN, HIGH);
  //delay(1000);
  digitalWrite(LED_BUILTIN, LOW);

}

void sendata(int16_t in, int16_t out){
  // encode int as bytes
  //byte payload[2];
  Serial.println(in);
  Serial.println(out);
  loraData[0] = highByte(in);  // left byte, lower value
  loraData[1] = lowByte(in);

  loraData[2] = highByte(out);
```

```
  loraData[3] = lowByte(out);

  Serial.println("Sending LoRa Data...");
  lora.sendData(loraData, sizeof(loraData), lora.frameCounter);
  Serial.print("Frame Counter: ");Serial.println(lora.frameCounter);
  lora.frameCounter++;
}


//Display current people count on 4-digit display

  //If the number of people is too high, trigger the buzzer



//Returns the distance of the ultrasonic sensor that is passed in
//a[0] = echo, a[1] = trig
int measureDistance(int a[]) {
  pinMode(a[1], OUTPUT);
  digitalWrite(a[1], LOW);
  delayMicroseconds(2);
  digitalWrite(a[1], HIGH);
  delayMicroseconds(10);
  digitalWrite(a[1], LOW);
  pinMode(a[0], INPUT);
  long duration = pulseIn(a[0], HIGH, 100000);
  return duration / 29 / 2;
}
```

3. Code TAGO.IO

https://github.com/bastvdn/festi.peuple