Réseaux longue portée, très basse consommation et bas débit Application pédagogique, supervision de batteries

Culture Sciences de l'Ingénieur

Anthony JUTON – Julien WARENGHEIM
Moez EZZERELLI – Fabien GUILLARD

Édité le
07/10/2019

école normale supérieure paris—saclay

Cette ressource est issue d'une publication de La Revue 3EI, numéro 96 d'avril 2019.
Anthony Juton est professeur agrégé à l'ENS Paris-Saclay, Julien Warenghem et Moez Ezzerelli sont ingénieurs Polytech Paris-Sud et Fabien Guillard, technicien SNCF à la gare de Rennes.

Cette ressource présente un exemple d'application LoRa développée par des étudiants de Polytech Paris-Sud, sur un cahier des charges de la SNCF. Elle fait suite à la ressource « *Réseaux très basse consommation, longue portée, bas débit, l'exemple de LoRaWAN* » [1].

Dans la gare SNCF de Rennes, le service d'accueil des personnes à mobilité réduite utilise quotidiennement des rampes d'accès PSH (Guldmann LP11) qui permettent d'installer des personnes en situation de handicap à bord des trains. Les rampes fonctionnent sur batterie et sont stockées sur les quais, sans possibilité de charge des batteries. Un technicien rapporte donc chaque semaine les batteries sur le lieu de charge le soir pour les replacer dans les rampes le matin (le long des 3 km cumulés de quais).

Pour optimiser l'utilisation des rampes, plusieurs critères sont à optimiser : le temps de maintenance, le taux de charge et la durée de vie des batteries.

Figure 1 : Rampe d'accès PSH Guldmann LP11

Début 2017, Fabien Guillard fait appel à l'IUT de Cachan / Polytech Paris-Sud pour développer une solution LoRa (la gare de Rennes ayant son propre réseau LoRa) qui permette de mesurer à distance le taux de charge des batteries. Julien Warenghem et Moez Ezzerelli, étudiants en école d'ingénieur à Polytech Paris-Sud ont travaillé sur ce cahier des charges pendant un an pour développer une preuve de faisabilité (POC, Proof of Concept).

La solution développée par ces étudiants, après analyse et développement par le service IoT de la SNCF, est en cours de développement et devrait passer en phase d'intégration. L'objectif à terme est de nationaliser le projet.

1 – Description fonctionnelle du système

L'objectif est de concevoir un système de supervision de batteries (2 batteries 12V / 7,2 Ah) pour des rampes d'accès pour personnes à mobilité réduite au sein d'une gare pour :

- Assurer le service d'accueil d'avoir une batterie chargée pour chaque rampe ;
- Diminuer le nombre de charges des batteries (et donc la charge des techniciens) en ne chargeant que les batteries ayant un niveau de charge inférieur à 70 % ;
- Augmenter la durée de vie des batteries en limitant le nombre de cycle de charge.

On trouve donc 2 fonctions principales :

- Déterminer le niveau de charge (SOC, State of Charge) et l'état de vétusté (SOH, State of Health) de la batterie ;
- Superviser de manière centralisée ce niveau de charge.

Une fonction de localisation de la batterie, un temps évoquée, a été rapidement abandonnée, le réseau LoRa n'offrant pas une précision suffisante, d'autant plus en intérieur, pour connaître sur quel quai est une batterie. Chaque batterie sera donc affectée à une rampe précise.

Le nombre de batteries de rampe étant important (14 rampes à la gare de Rennes), la partie matérielle installée sur la batterie doit coûter quelques dizaines d'euros maximum, en lien avec le service qu'il apporte. Le réseau LoRa existant à la gare de Rennes, le reste des coûts est du développement matériel et logiciel, pour lequel la SNCF investit dans son effort de profiter des objets connectés pour améliorer gares et infrastructures.

Le système ne doit pas participer significativement à vider la batterie. Les batteries étant chargées au moins une fois par mois, une consommation moyenne inférieure à 24 mW est visée (autonomie théorique de 10 mois du système sur une batterie inutilisée).

Le système de mesure et de transmission ne doit pas être intrusif, pour ne pas menacer la garantie du matériel.

2 – Le système de mesure embarqué

Le système de mesure et de transmission de l'état de la batterie doit être placé dans le boîtier plastique regroupant les 2 batteries 12V et la connectique.

2.1 - Architecture fonctionnelle du système

L'état de charge (State of Charge SOC) de la batterie s'obtient par une mesure de la tension à vide, via un pont diviseur de tension de grande valeur.

L'état de vieillissement (State of Health SOH) de la batterie s'obtient par un essai en charge qui permet de déduire la résistance interne de la batterie. Une décharge à 1 A, (dans une résistance de 20 ohms) pendant 10 ms est choisie pour cet essai.

Un module Multitech *mdot* utilisant l'environnement de programmation orienté « objets connectés » *mbed* est choisi pour la transmission des informations en LoRa. L'environnement *mbed* propose de nombreux exemples pour relier ce module *Multitech mdot* à la passerelle *Multitech Multiconnect Conduit* que nous verrons dans le chapitre suivant.

L'architecture suivante est alors retenue pour le système de mesure de l'état de la batterie :

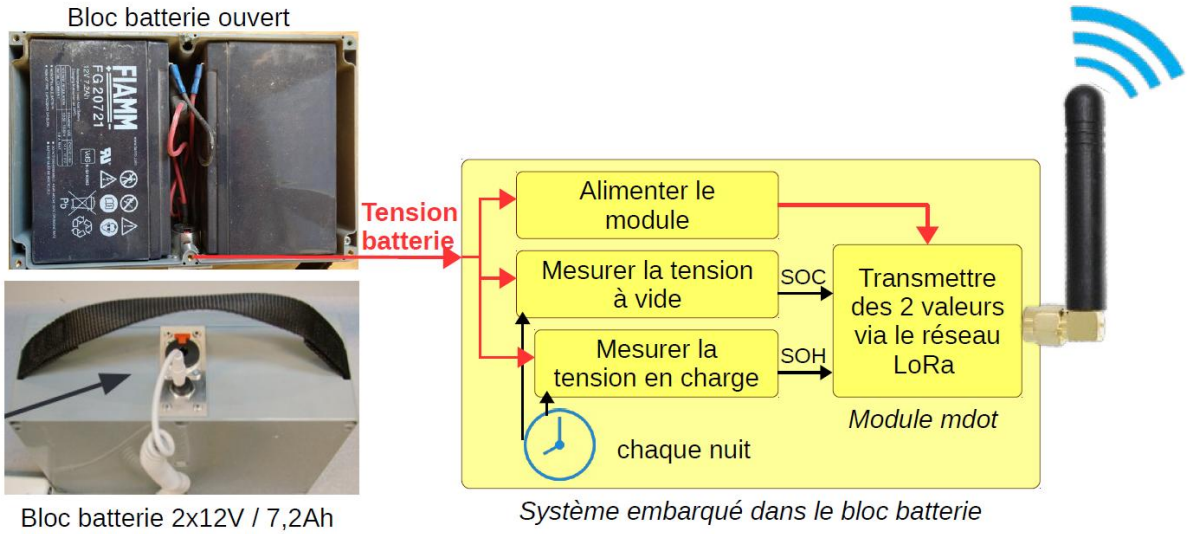


Figure 2 : Architecture fonctionnelle retenue pour le système de mesure embarqué dans le bloc batterie

2.2 - Réalisation du système de mesure embarqué

Le pont diviseur de tension utilise des résistances de 47 et 7 kiloohms, de sorte de limiter le courant à 500 µA. On aurait pu envisager, comme pour la résistance de charge de déconnecter ce pont diviseur, hors mesure.

La mesure de la tension en charge de la batterie se fait en fermant 10 ms un transistor NMOS connectant la batterie à une résistance de 20 ohms. La consommation est alors proche de 1 A, pendant 10 ms, une fois par jour, soit une moyenne de 0,12 µA.

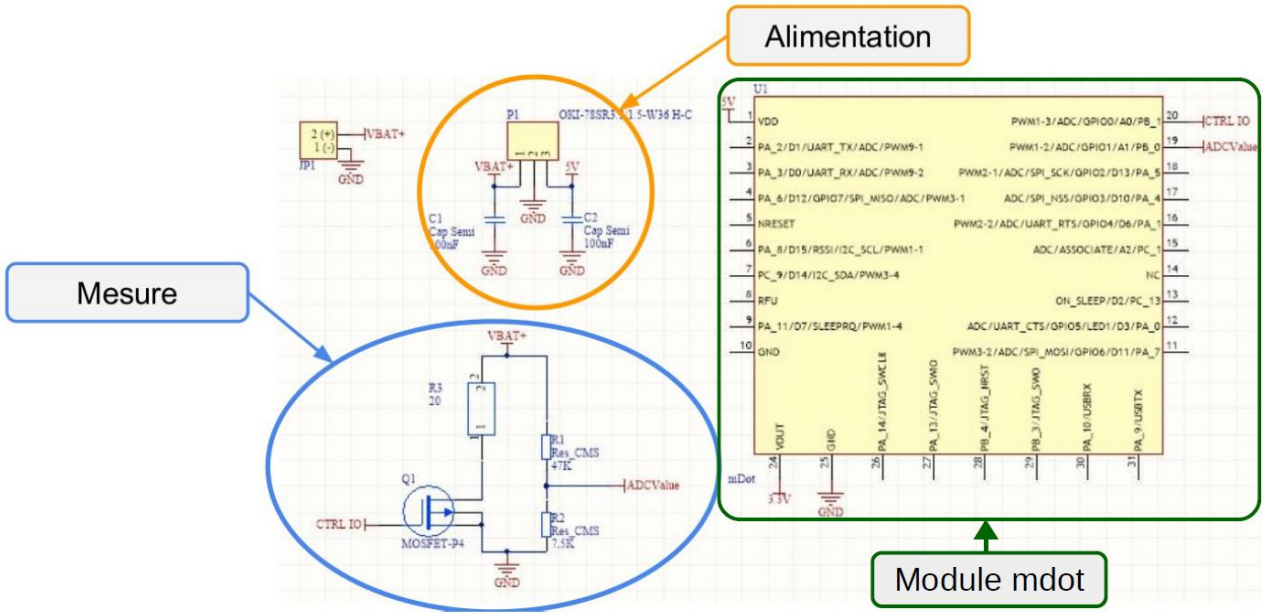


Figure 3 : Schéma électrique du système de mesure embarqué

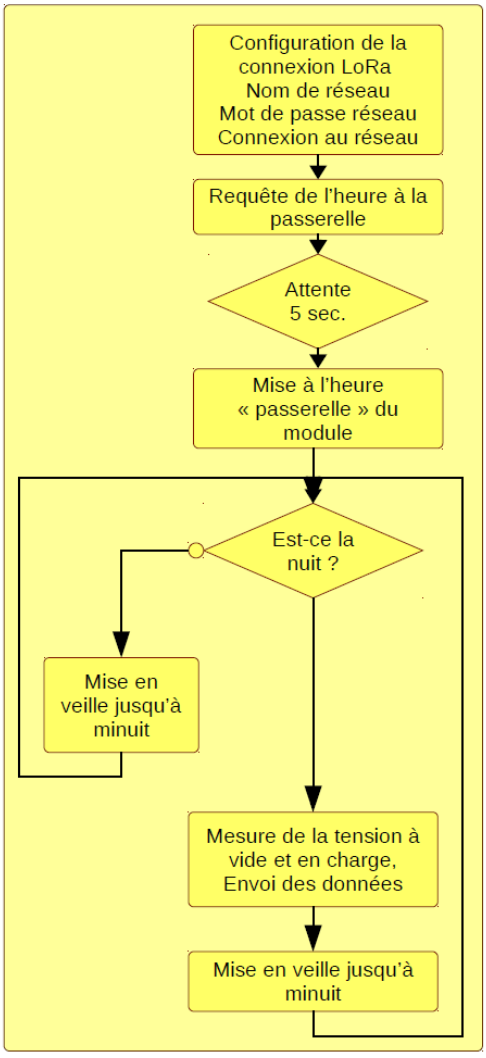
Le module Multitech mdot est composé d'un microcontrôleur STM32F411 et d'un tranceiver 868 MHZ Semtech SX1272. Le module annonce une consommation en veille inférieure à 50 µA mais 200 mA en transmission.

Un travail resterait à faire sur la conversion 24V → 5V, le régulateur à découpage choisi ayant une consommation certaine (5 mA) à vide.

Figure 4 : Module LoRa Multitech mdot

2.3 - Programmation du module mdot

Le module mdot dispose de la possibilité de passer en mode sleep (consommation annoncée 40 µA) avec un réveil sur minuterie. On fait en sorte que le module se réveille à minuit, fasse ses deux mesures SOC et SOH et envoie les valeurs au serveur, avant de se mettre en veille de nouveau jusqu'à minuit.




```
static std::string network_name = "MTCDT-87654321";
static std::string network_passphrase = "12345678";
...
update_ota_config_name_phrase(network_name, network_passphrase,
...
join_network();


send_data(tx_request_time);


wait(5);


ret = dot->recv(rx_recv_time);



if ((current_hour >= 0) && (current_hour<=3)) {
    // IF YES : GO TO COMPUTE
    actual_state = state_compute;
} else {
    // IF NO : GO TO SLEEP
    actual_state = state_sleep;

    hours2midnight = 24-current_hour;
    sleep_duration_s = hours2midnight*3600; // wake u
}
// GO TO SLEEP
dot->sleep(sleep_duration_s, mDot::RTC_ALARM, false);

float soc;
soc = getSOC();
float soh;
soh = getSOH();
// SEND SOC AND SOH TO GATEWAY

// GO TO SLEEP
actual_state = state_sleep;
```

Figure 5 : Algorithme du programme du mdot avec quelques extraits de code

3 – La chaîne d'acquisition et de diffusion de l'information

3.1 - Architecture retenue pour le système d'acquisition

Plutôt que de créer un serveur LoRa, tâche ambitieuse pour les novices en LoRa que nous étions, il est choisi d'utiliser une passerelle Multitech Multiconnect conduit (qui fait à la fois passerelle et serveur réseau LoRa). Celle-ci, configurable depuis son serveur web, permet assez aisément de

mettre en place la réception des messages LoRa et l'émission d'informations via des messages MQTT, protocole conçu pour les transmissions issues d'objets connectés.

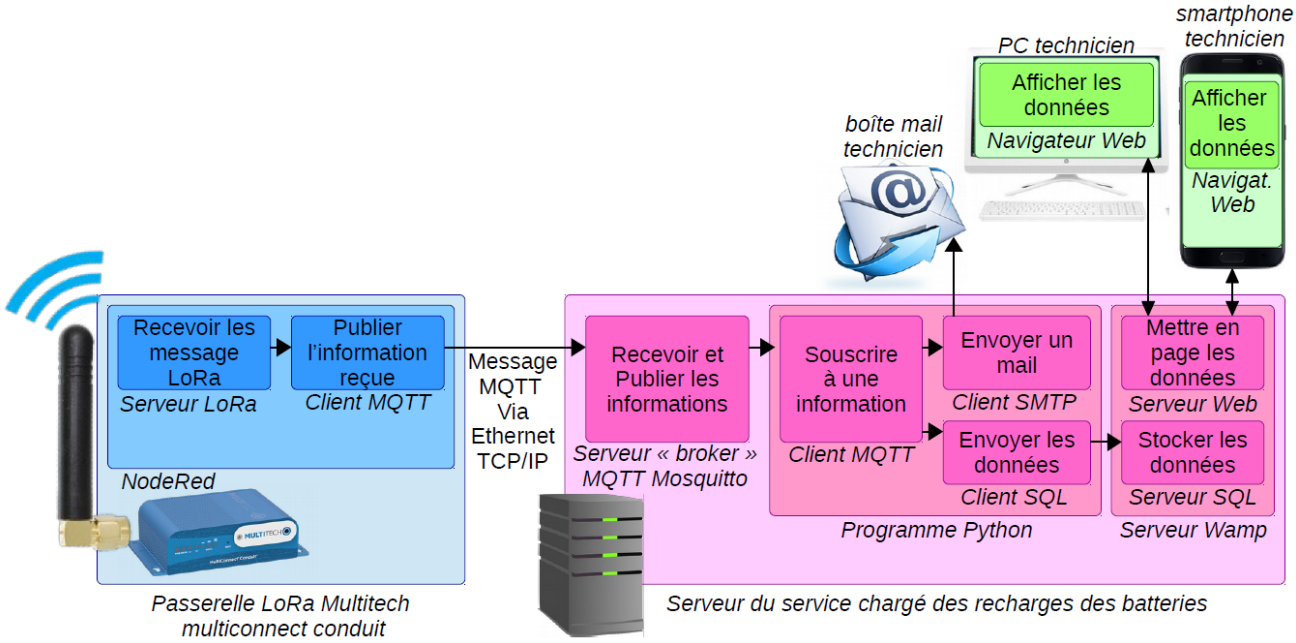


Figure 6 : Architecture fonctionnelle retenue pour la chaîne d'acquisition des données

Un serveur MQTT est installé sur un PC, représentant le serveur du service chargé des batteries. Un programme python récupère les informations en MQTT, envoie un mail une fois par jour au technicien pour indiquer les batteries à charger et remplit une base de données *mySQL* historisant les informations des batteries. Un serveur web donne accès aux données de cette base.

3.2 - Configuration / programmation de la passerelle

La passerelle *Multitech Multiconnect Conduit* dispose d'un serveur web embarqué pour sa configuration.

Figure 7 : Passerelle Multitech

Figure 8 : Kit de démarrage Multitech : 1 passerelle Conduit + 2 modules mdot

On y trouve la configuration du réseau LoRa et un logiciel Node-Red permettant de programmer de manière graphique le comportement du serveur LoRa.

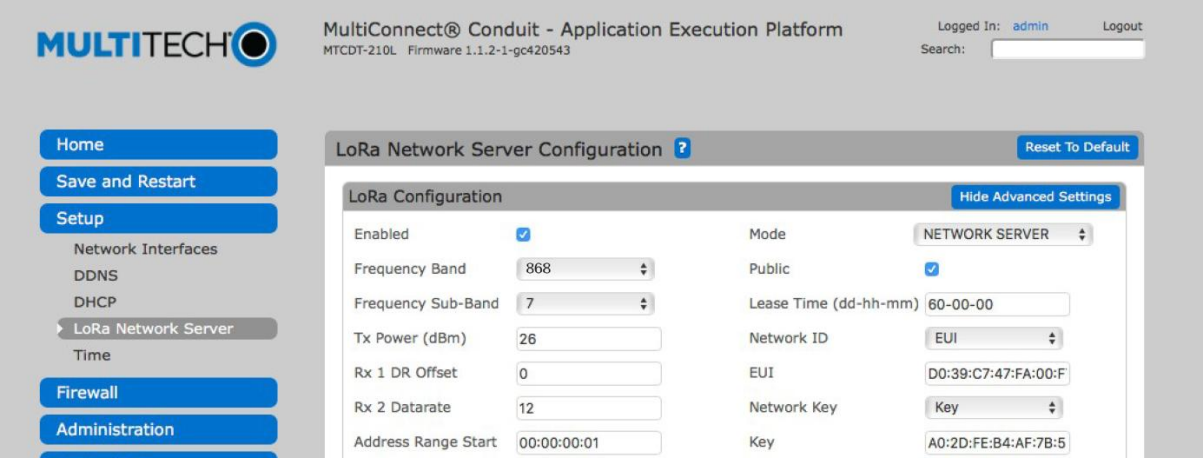


Figure 9 : Ecran de configuration du réseau LoRa de la passerelle Multitech

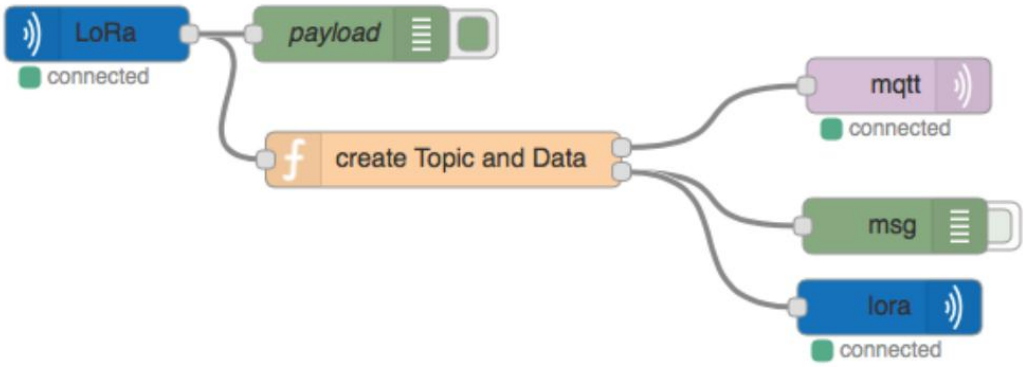


Figure 10 : Schéma du programme Node-Red implanté dans la passerelle Multitech

Le schéma ci-dessus indique que lorsqu'une trame est reçue, son contenu (payload) est affiché localement dans la fenêtre de debug d'une part. D'autre part, son contenu est traité par la fonction « Create Topic and Data » :

```
// If time is detected send time through string
if(msg.payload == "time"){
      //console.log(datejour);
      //msg.payload = { payload : Date.now().toString()  };
      msg.payload = Date.now().toString();
      msg.ack = false;
      return [,msg]
}else{

    var topic2 = "device/"+msg.eui;
    var newmsg = { topic: topic2, payload : msg.payload  };
    return [newmsg,]
}
```

Figure 11 : Code de la fonction Create Topic And Data

Si le message LoRa reçu par la passerelle est « time », la passerelle renvoie la date et l'heure au module demandeur. Sinon, la passerelle prépare un message MQTT avec le contenu du message LoRa reçu.

3.3 - Le broker MQTT sur le serveur

MQTT est un protocole application de TCP/IP, populaire dans les communications liées aux objets connectés. Très léger, il est peu gourmand en énergie et en bande passante, mais il permet tout de même un fonctionnement sécurisé.

Il fonctionne sur le mode souscription / publication. Les clients (notre passerelle *Multitech* et notre programme Python) souscrivent à un même sujet / topic « device/ » suivi de l'identifiant du module *mdot*.

- Quand la passerelle envoie un message au serveur/broker MQTT, ce dernier conserve le message jusqu'à ce qu'un nouveau message avec ce même sujet / topic arrive ;
- Quand le programme Python demande au serveur / broker MQTT si des nouvelles sont arrivées sur les topics auxquels il a souscrit, on lui répond avec la nouvelle valeur arrivée.

Un serveur (appelé « broker ») mosquitto (serveur MQTT open source) est installé sur la machine. Il reçoit les messages MQTT envoyés par la passerelle Multitech via le réseau Ethernet TCP/IP.

3.4 - Le programme Python sur le serveur

Le programme est écrit en Python, pour sa portabilité et pour la richesse des bibliothèques disponibles : Client MQTT, Client SQL, Client SMTP.

On crée un client MQTT abonné au topic "device/#" le "#" indique n'importe quelle donnée se trouvant à la suite de "device/"

```
######### DEFINE

#MQTT
TOPIC = 'device/#'
BROKER = "localhost"
PORTMQTT = 1883
```

Ainsi on récupère la totalité des modules de mesure de tension. Le client MQTT boucle en continu par la suite et si une déconnexion a lieu il tente de se reconnecter.

```
############ MQTT

#Create an MQTT Client
client = mqtt.Client()

#Connect to the on_connect fucntion as a callback
client.on_connect = on_connect

#Connect to the on_message fucntion as a callback
client.on_message = on_message

# define the connection server
client.connect(BROKER, PORTMQTT, 60)

# infinite Loop
client.loop_forever(timeout=1.0, max_packets=1, retry_first_connection=False)
#client.loop_start()
```

On associe la fonction callback "on_message", qui lors de la réception d'un message se déclenche. Cette fonction récupère les informations du message.

```
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):

	capteur_name = msg.topic.split("/",2);

	#print(capteur_name[1])

	if int(msg.payload) < 70 :
	envoi_mail(msg, capteur_name[1])

sql = "INSERT INTO data_capteur (ID, nomcapteur, valeur, date) VALUES (NULL, %s,
%s, CURRENT_TIMESTAMP)"

	with conn.cursor() as cursor:
	cursor.execute(sql,(capteur_name[1],msg.payload))
	conn.commit()

	print(msg.topic+" "+str(msg.payload))
```

Puis celle-ci enregistre les messages entrants dans la base de données SQL en fonction du capteur et de sa donnée (la date et l'index sont automatiquement mis à jour lors de la réception de nouvelles informations dans la base de données).

Dans la fonction "on_message", on ajoute une fonction "envoi_mail" qui si la donnée de niveau de batterie est inférieure à 70 %, envoie un mail à une ou plusieurs personnes pour leur signaler le niveau de tension ainsi que le nom de la batterie à recharger.

```
#Mail
def envoi_mail(msg, name):
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login("julien.warenghem@gmail.com", "Chaussette91")
val = "Battery " + name + " under 70% (level = " + msg.payload + " %)."
server.sendmail("julien.warenghem@gmail.com", "julien.warenghem@u-psud.fr", val)
server.sendmail("julien.warenghem@gmail.com", "moez.ezzerelli@u-psud.fr", val)
server.quit()
print(val)
```

3.5 - Le serveur Wamp (serveur SQL et serveur Web)

Le serveur Wamp est bien connu des éditeurs de site web. Il dispose d'une base de données mySQL et d'un compilateur PHP permettant de faire des pages web dynamique.

Une base de données mySQL est donc créée pour stocker les données des différents capteurs.

Pour afficher les données, l'outil PHP "JpGraph" permet de concevoir des graphiques rapidement.

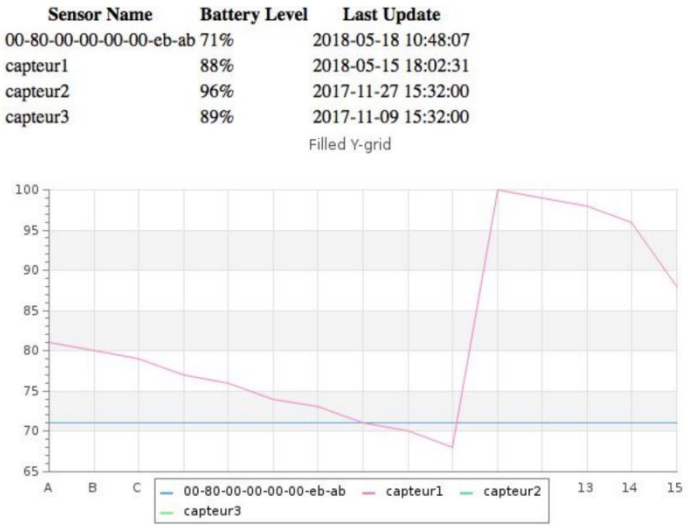


Figure 12 : Affichage de l'évolution de la charge d'une batterie sur la page web

4 – Conclusion

À travers ce projet basé sur un cahier des charges concret, les étudiants ont pu démontrer la faisabilité d'une solution d'acquisition utilisant un réseau LoRa.

Ils ont su mettre en service un réseau local LoRaWan à partir d'une passerelle et de modules Multitech. Ils ont exploité le protocole MQTT pour transmettre les messages sur le réseau TCP/IP et ont stocké les informations dans une base de données.

Un peu juste en temps, ils n'ont pas pu tester la portée annoncée du module (2 km en champ libre), ils n'ont pas mesuré la consommation effective en mode veille et n'ont pu faire de statistiques sur la disponibilité du réseau.

Une fois intégré, le système, si il devient national, permettra à la SNCF, en plus d'améliorer la charge de ses batteries, de mettre en place un suivi et une analyse d'une grande quantité de données batterie. On peut espérer ainsi optimiser le cycle de vie de l'ensemble des batteries, peut-être prévoir les remplacements, mesurer quantitativement les effets du stockage à l'extérieur, comparer des marques de batteries sur le long terme…

Références :

[1]: Réseaux très basse consommation, longue portée, bas débit, l'exemple de LoRaWAN,
A. Juton, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/reseau-tres-basse-consommation-longue-portee-bas-debit-exemple-lorawan