MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering

2021 – 2022

Internet of Things

Lab 3 : Alarme pour vélo

Anuraag POTHULA

anuraag.pothula@master.hes-so.ch

HES-SO MSE, 24/12/2021

Introduction

Pour ce projet le but était de développer une solution de traqueur de vélo low-power et de pouvoir activer et désactiver l'alarme à distance.

Le device est une carte pycom lopy 4 monté sur un shield pytrack. Celui-ci permet de localiser par GPS/GNSS la position du vélo, détecter du mouvement et s'il le faut lancer l'alarme. Ce device communique via LoRa avec un nano gateway connecté à The Things Network (The Things Stack).

La solution idéale :

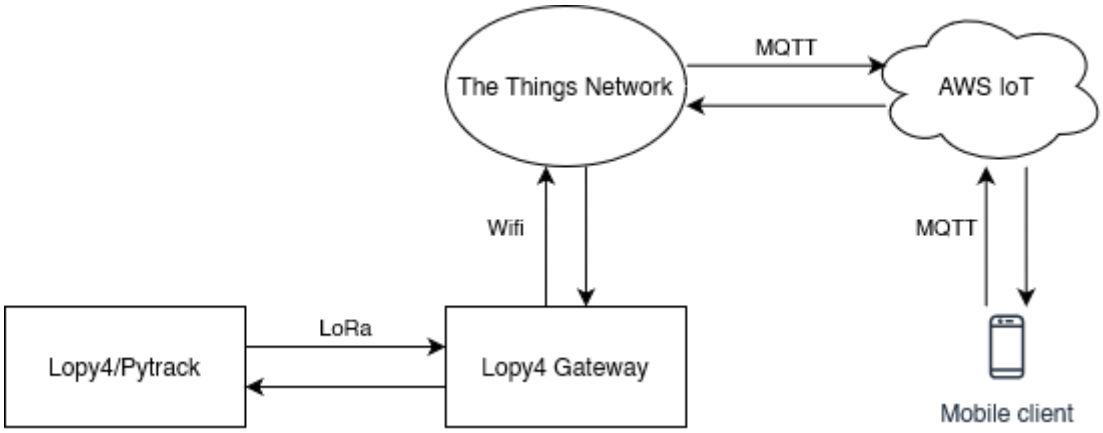

Figure 1 : architecture d'une solution optimale

Ce que nous avons :

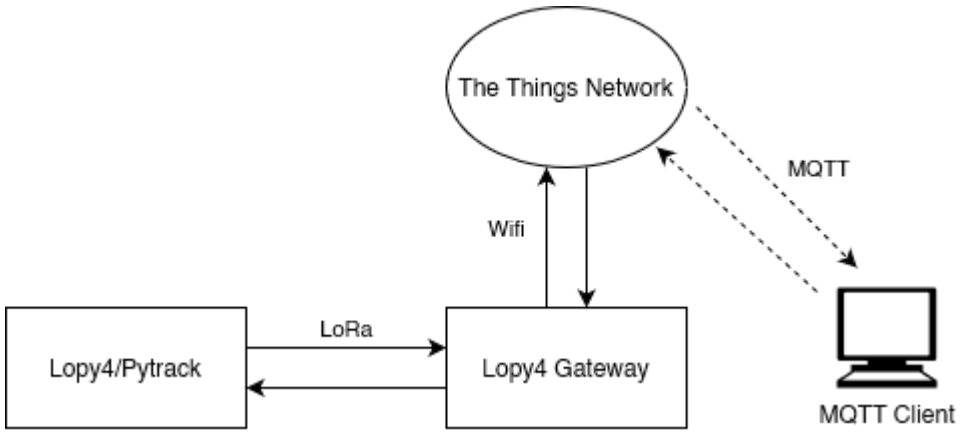

Figure 2 : architecture de ce que nous avons pu faire jusqu'à maintenant

Lopy/Pytrack node

Ici nous avons pu utiliser le shield Pytrack et son module L76GNSS pour récolter les données de localisation. Nous avons mis en place une machine d'état pour les différents states du device lopy/pytrack. La LED RGB a aussi été programmé pour permettre de visualiser l'état du device.

Nous commençons toujours en *Idle*. La fonction *set_state()* va faire passer l'état en *Locking* ou *Idle* ou ne changera pas l'état dépendant des messages qu'il reçoit. Dans l'état *Locking*, nous enregistrons la position lors de la mise sous alarme. Ainsi nous pourrons comparer celle-ci avec la nouvelle position lorsque le device est dans l'état *Locked*. En *Locked*, le device envoi sa position toutes les 5 minutes (similaire à *Idle*). Quand la différence de latitude/longitude est supérieure à 0.0003 degrés (environ 30m), l'état *Danger* est activé. Là nous auront une transmission toutes les 30 secondes jusqu'à ce qu'un message de désactivation de l'alarme soit reçu. Évidemment pour capter des données GPS/GNSS, il

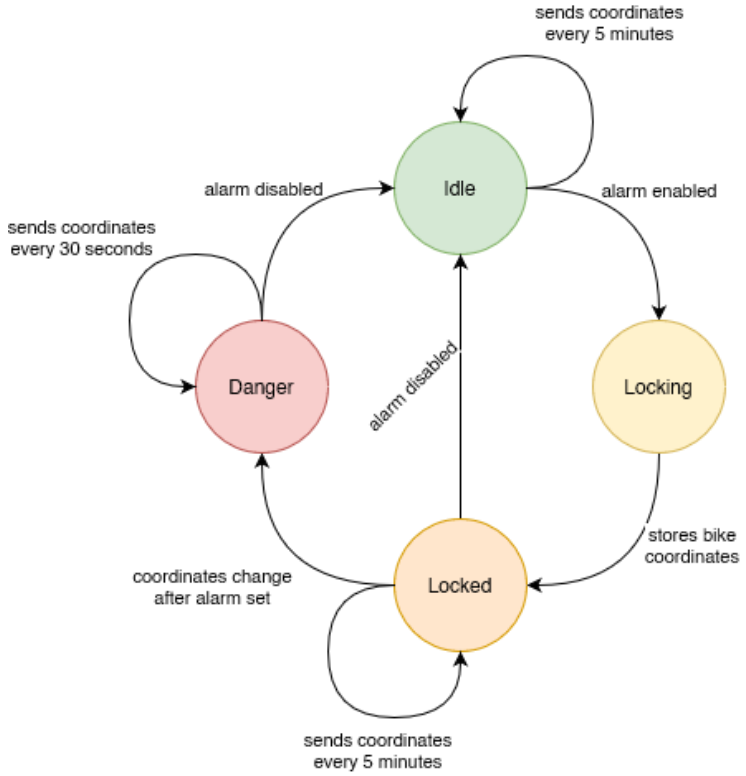

Figure 3 : Machine d'état du device

faut que le device soit à l'extérieur.

Lors de la démo les délais de transmission de 5 min et 30 secondes ont été réduit à 5 secondes par intérêt de praticité, et l'alarme passe à l'état danger lorsque les coordonnées GPS/GNSS sont *None* (pas dû à la distance parce que nous ne pouvons pas actuellement nous déplacer avec le module alimenté).

Lopy Gateway/TTN

Sans couverture de la part de *The Things Network*, nous avions besoin d'un gateway pour nous connecter à leur stack. Un lopy avec une carte d'expansion a été configuré pour se connecter au serveur de TTN via un réseau WLAN local. Ce nano-gateway LoRaWAN prends des paramètres en accords avec les normes loRa EU868.

Sur l'interface TTN nous avons donc pu ajouter un gateway en fournissant le *Gateway_ID* et le reste des informations nécessaire avant de le connecter. Après avoir crée une application via leur interface nous avons pu ajouter un end device et l'associer à notre device physique via le DevEUI. Ensuite nous avons pu activer le device et joindre de réseau pour enfin envoyer nos donné de positionnement.

Les données sont envoyées par paquet de 9 bytes : 4 bytes pour la longitude, 4 bytes pour la latitude et 1 byte pour valider les données. Le byte de validité sert à indiquer si la valeur des coordonnées est correcte ou si nous n'avons pas pu avoir de signal. Sur l'interface TTN, il a fallu implémenter un formateur de payload pour décoder correctement les données.

MQTT

Nous avons tenté d'abords tenté une intégration avec AWS IoT mais sans autorisation pour déployer AWS CloudFormation nous nous sommes repliés sur une intégration avec un simple client MQTT. Ici nous n'avons réussi à nous connecter au serveur TTN en TLS et envoyer des données en downlink pour activer et desactiver l'alarme du device.

Conclusion

Ce projet est encore loin d'être complet. Nous arrivons à obtenir les données de positionnement et les envoyer par LoRa à TTN. Mais une intégration de AWS aurait permis une meilleure gestion des devices avec une solution plus scalable. Passer par un service de cloud computing (peut-être avec AWS EC2) nous aurait permis de traiter certaines données ou manque de données en uplink même lorsque notre client MQTT n'est pas actif, par exemple pour rapidement alerter l'utilisateur si nous ne recevons plus de données pendant une longue période. Certaines parties du codebase auraient pu être mieux découplées et documentées. Enfin expérimenter avec les fonction de deepsleep entre les communication pourrait se révéler utile pour prolonger le cycle de vie avec une seule batterie.

Par manque de temps et de ressources nous n'avons pas pu aller plus loin dans ce projet, mais évidemment une intégration du client MQTT dans une application android avec kotlin/java aurait aussi été plus judicieuse.