Rapport

CampusIoT

WEILL William

BESNIER Benjamin

BESNARD Guillaume

DEPRIESTER Timothée

LEVESQUE Théo

Encadrant : Didier Donsez

Table des matières

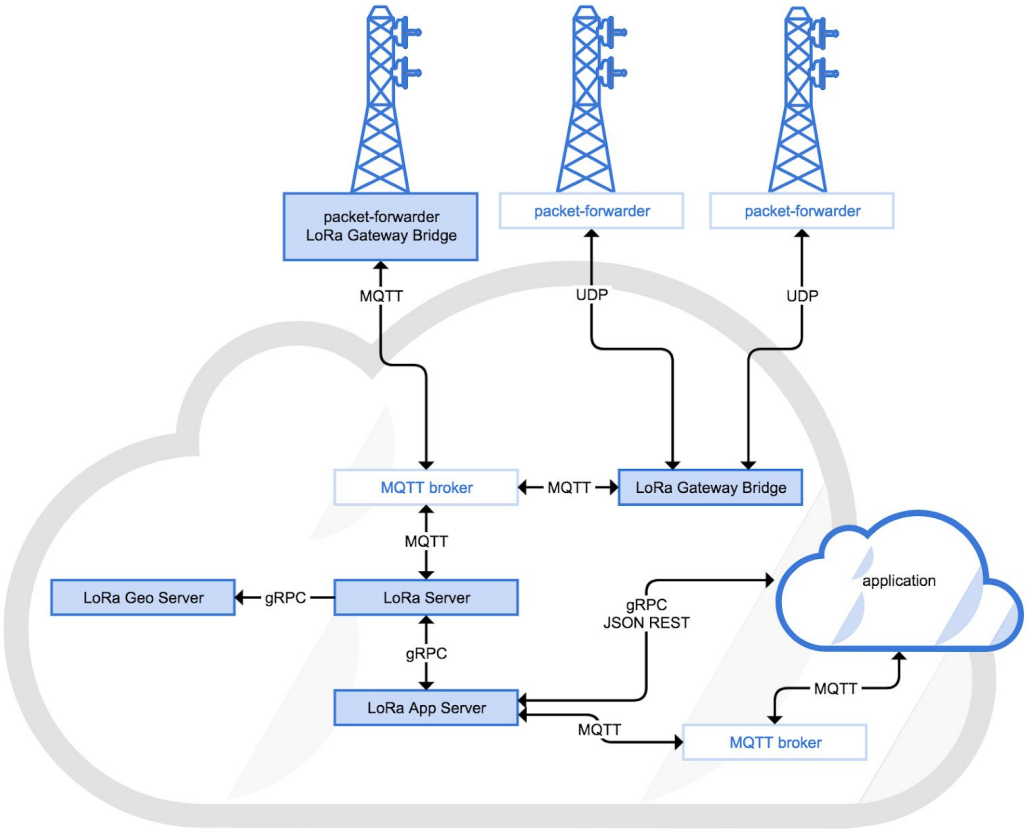
1. Rappel du sujet/besoin et cahier des charges

Le projet consiste à l'ajout de nouvelles fonctionnalités pour le projet sources ouvertes LoRa Server, dans le but d'améliorer la plateforme CampusIoT. Ces fonctionnalités comprennent : l'ajout d'authentification de passerelles réseaux par jeton ; la création d'une API de géolocalisation pour s'abstraire de l'API Collos ; l'ajout d'une carte sur l'interface graphique pour faciliter la visualisation de la localisation des passerelles ; la gestion du *duty cycle* des passerelles pour les liaisons descendantes au niveau serveur.

2. Technologies employées

LoRa Server est un serveur de réseau et d'application LoRaWAN aux sources libres codé en Go. Il se compose de plusieurs briques logicielles communiquant en utilisant la librairie d'appel de fonctions à distance basée sur protobuf, gRPC. L'utilitaire Gateway-gRPC est utilisé pour générer la base du code Go ainsi que les descriptions Swagger de l'API de la partie web du projet. Avec cela est utilisée une base de données relationnelle PostgreSQL ainsi qu'un courtier MQTT, ici Mosquitto. Pour permettre une authentification plus fine, nous avons ajouté un greffon à Mosquitto, go-auth-plug. Ce dernier permet une authentification depuis plusieurs sources, dont une base de données PostgreSQL.

3. Architecture techniques



4. Réalisations techniques

a. Algorithme géolocalisation

L'algorithme de géolocalisation réalisé se divise en deux parties : la première utilise le RSSI (Received Signal Strength Indication) obtenu sur 3 passerelles, mais cet algorithme présente un problème auquel aucune solution simple est envisageable : il faut convertir ce RSSI obtenue en distance. Or, pour cette conversion nous avons besoin de la puissance d'émission de l'objet voulant être localisé, puissance qui est propre à chaque objet. Cette puissance d'émission n'est pas donnée dans le paquet renvoyé par les passerelles elle doit donc être précisée directement dans le code, cette solution ne peut donc pas être mise en production. Le deuxième algorithme utilise quant à lui les TOA renvoyés par les passerelles LoRa. Cette deuxième version a cependant besoin de 4 passerelles LoRa pour trouver la

position de l'objet mais est grandement plus précise que la précédente puisque la conversion de temps en distance, grâce à la célérité de l'onde, est aisée et précise tandis que celle de RSSI à distance n'est qu'une approximation. Néanmoins, cet algorithme n'est pas exact pour autant, en effet physiquement la Terre n'est pas une sphère mais s'approche plutôt d'une ellipse, en plus de cette approximation s'ajoutent des sources d'erreurs d'ordre technique dont l'imprécision des TOA renvoyés par certaines gateways et la troncature mathématiques de certains résultats de calculs qui donne donc une position pas tout à fait exacte.

b. Changement backend géolocalisation

Le choix du *backend* de géolocalisation est rendu modulaire. Le choix s'effectue dans la configuration *lora-geo-server.toml*. Ce choix permettra de changer l'api utilisée pour les tests et pour la production. Il n'y a pas de clé d'authentification pour notre api à la différence de celle de Collos.

Dans le *lora-geo-server* et *loraserver,* des tests ont été enlevées pour que toutes les passerelles puissent maintenant utiliser la géolocalisation, et pas seulement celle disposant d'un *FineTimestamp*. Il faut pour l'instant un minimum de 3 passerelles, mais il faudra monter ce minimum à 4 pour l'utilisation de notre nouvelle implémentation de l'algorithme de localisation qui prend en compte la rotondité de la Terre.

c. Authentification MQTT

La transmission des paquets de données entre les passerelles LoRa et le LoRaServer est assuré par un courtier MQTT, ici présentement Mosquitto. Les flux de données peuvent être sécurisés mais cela implique l'utilisation d'un système d'authentification. De base, Mosquitto ne permet qu'une authentification très limitée, qui consiste à inscrire les couples nom d'utilisateur / mot de passe dans un fichier texte. Afin de rendre le processus dynamique, nous avons utilisé un greffon intitulé *go-auth* qui permet d'utiliser de nombreuses sources différentes, notamment *PostgreSQL*, le système de gestion de base de données exploité par LoRaServer. Pour effectuer la génération de la clé d'authentification, nous avons dû modifier le serveur d'application (lora-app-server) afin d'ajouter un champ *mqtt_key* dans la base de donnée ainsi que l'API, afin de permettre l'affichage et la modification de la clé, et aussi l'interface web pour l'interaction avec l'utilisateur.

d. Carte Leaflet

Lors de la réception d'un paquet d'un objet, s'il n'y a pas assez de passerelles ayant reçu le message, les informations de réception sont transmises au geo-api qui calcule la position approximative de l'objet et la retransmet au LoRaServer pour le garder en base de données.
Nous avons créé un nouvel onglet sur la page de chaque objet enregistré en React, récupérant la localisation de l'objet stocké en base de données et transmise par l'API, puis l'affichant sur une carte Leaflet.

5. Mise en place environnement de développement et de tests

Afin de faciliter le développement des différentes briques logicielles, nous utilisons des techniques d'intégrations continues. Les différents services sont conteneurisés grâce à Docker, ce qui facilite le déploiement. Afin que toutes les personnes du projets puissent obtenirs la dernière versions des différentes briques logicielles, nous avons mis en place une construction automatique des images de conteneurs lors de chaques modifications de la branche *master* de chaque brique logiciel. Pour réaliser ceci et faciliter la distribution des images, nous avons utilisé le service de dépôt d'image Docker, Docker Hub[1], qui permet la construction automatique depuis GitHub et la mise à disposition des images dans le nuage.

6. Gestion de projet

La répartition des rôles des membres du groupes :

- William Weill - Chef de projet, Développement (Algorithme Géolocalisation)
- Timothée Depriester - DevOps, Recherche problématique Duty Cycle
- Benjamin Besnier - Développement (Algorithme Géolocalisation), dactylographe
- Guillaume Besnard - Opération, Intégration, Test

[1] https://hub.docker.com/

- Théo Lévesque - Opération, Développement authentification ACL, Token, Intégration, Test

La planification s'est déroulée en deux parties, dans un premier temps nous avons dû nous accorder deux journées afin de cerner le but du projet et donc pouvoir établir le planning effectif du développement qui constitue donc la deuxième étape de la planification. Malgré cette stratégie visant à établir une planification la plus réaliste possible, entre les désagréments venant de technologies que nous ne maîtrisons pas suffisamment et les clarifications nécessaires sur le rendu attendu par le client, nous n'avons pu nous tenir strictement à ce qui était prévu.

Au début du projet, les différentes missions que nous devions réaliser tout au long du projet n'étaient pas définies. Nous avons eu seulement quelques petites taches, et, au cours du projet et des différentes discussions et réunions avec notre client, nous avons pu définir des éléments à réaliser.

Après la phase d'analyse et de prise en main du système, lorsque nous avons voulu ajouter des fonctionnalités, nous nous sommes rendu compte d'un problème logiciel bloquant toutes modifications en profondeur du code. Nous avons alors tenté de corriger le problème pendant plusieurs jours en vain mais finalement, seulement le développeur du système original avait la capacité de résoudre ce problème. C'est alors une semaine plus tard que le problème fut résolu et que nous avons réellement pu réaliser les tâches fondamentales de notre projet.

Dans l'ensemble nous aurions dû plus nous tenir à la ligne de conduite des méthodes agiles puisque notre client a fait évoluer le projet au cours de l'avancement et, dans ce cas, les deux jours que nous avions consacrés à l'établissement d'un planning réaliste se sont montrés inefficaces.

7. Métriques

Notre API de géolocalisation est de type REST sans-état, donc on peut très facilement la mettre à l'échelle et supporter la charge.

En calculant le temps ingénieur passé sur le projet, nous arrivons au résultat de 180 heures ; En effet, nous avions 4 étudiants à temps pleins et 1 à un temps partiel (4 jours sur 5) car il suit le cursus IAE.

8. Conclusion

Un des problèmes qui nous a le plus impacté est le fait de travailler sur un logiciel aux sources libres et donc de dépendre très fortement d'une personne extérieure avec très peu de moyen de communication avec cette dernière. Lorsqu'un problème surgit, s'il nous est impossible de le résoudre par nous-même, nous nous retrouvons dans l'incapacité d'avancer. Il a donc fallu trouver un axe de travail totalement disjoint de toutes les réflexions effectuées depuis le début du projet.

Nonobstant cet écueil qui nous a empêché d'avoir tous les résultats souhaités et envisagés, nous avons pu apprendre l'utilisation de nouvelles technologies et techniques (architecture microservice, intégration continue) ainsi que de nous apporter de nouvelles pistes de progressions sur le plan de la communication et de la gestion de projet.

9. Bibliographie

Organisation et dépots GitHub : https://github.com/campus-iot/
Dépôt des images Docker : https://hub.docker.com/u/theo024
Trilateration avec rssi : https://appelsiini.net/2017/trilateration-with-n-points/