Compte rendu TP2 IOT
Géolocalisation à partir du WiFi Sniffing avec ESP32

Grégoire MAHON
Armand LELONG
Polytech Sorbonne EI2I-4 II (Groupe B)

Table des matières

Introduction

Ce TP2 d'IoT, intitulé "Géolocalisation sans GPS : WiFi Sniffing", nous a permis de mettre en pratique des compétences en ingénierie réseau et en programmation. L'objectif était de développer un système capable de localiser un appareil sans utiliser le GPS, en exploitant les signaux Wi-Fi environnants.
Pour ce faire, nous avons utilisé un ESP32 pour scanner les réseaux Wi-Fi, puis envoyé ces données à Node-red pour traitement.
Nous avons ensuite formaté ces données et les avons envoyées via une requête GET à notre serveur HTTP Node-red, avant de les afficher sur une carte.
Les données de localisation des différents réseaux Wi-Fi nous ont été fournies sous forme de fichier CSV par notre chargé de TP.

Scanner des Wi-Fi avec l'ESP32 et Envoyer la liste des Wi-Fi sur Node-red

Dans cette partie du projet, notre objectif était de configurer l'ESP32 pour scanner les réseaux Wi-Fi environnants et d'envoyer des informations sur ces réseaux (SSID, RSSI, BSSID) à notre serveur Node-red afin de les traiter dans la partie suivante. Nous avons programmé l'ESP32 avec l'IDE Arduino, utilisant les bibliothèques *WiFi.h* et *HTTPClient.h* pour établir la connexion Wi-Fi et envoyer les données.

Le processus se déroule comme suit (cf code source dans /src/ get_wifis_ESP32/get_wifis_ESP32.ino) :

- **Connexion Wi-Fi** : L'ESP32 se connecte à un réseau Wi-Fi spécifique, avec un SSID et un mot de passe prédéfinis.

- **Scanning des Réseaux** : Ensuite, l'ESP32 scanne les réseaux Wi-Fi disponibles, enregistrant des informations comme le SSID, le RSSI (indicateur de force du signal reçu) et l'adresse BSSID de chaque réseau, cela permet par la suite de traiter ces informations pour connaître, à partir d'une base de données (fournie ici par notre chargé de TP), la position de notre ESP32 en fonction du RSSI des réseaux scannés etc.
- **Création du JSON** : Les données de chaque réseau sont formatées en JSON, une structure de données facilement interprétable par le serveur (simple boucle *for* pour mettre les données sous forme de JSON).
- **Envoi des Données** : Les données JSON sont ensuite envoyées au serveur Node-redvia une requête POST HTTP. Nous avons utilisé l'adresse IP du serveur Node-redet l'endpoint spécifique côté Node-red.

Ce processus est automatisé pour se répéter toutes les 10 secondes, permettant un scan régulier des réseaux environnants.

Le formatage des données au format JSON nous a permis par la suite de très facilement pouvoir récupérer les données depuis notre application Python afin de les traiter pour calculer la position de notre appareil.

Envoyer une requête GET au serveur HTTP

Par la suite, l'objectif était d'envoyer une requête depuis le serveur Node-redafin de récupérer la localisation de l'ESP32, en envoyant donc, avec la requête, les informations des réseaux WiFi scannés par l'ESP32.
Le serveur Node-red est donc configuré pour envoyer une requête GET au serveur Flask.

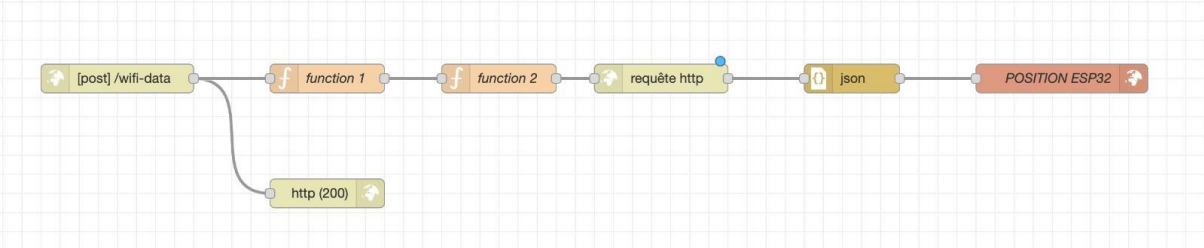

Figure 1 : Flow Node-red

Le serveur Flask est hébergé par notre application Python, fournie partiellement avec les documents du TP. Notre rôle a été de coder la réception de la requête GET et de coder « l'algorithme » de calcul de la position de notre ESP32.

Voici les détails de cette étape :

- **Transmission des données de l'ESP32 à Node-red** : L'ESP32 scanne les réseaux Wi-Fi et envoie les données collectées (SSID, RSSI, BSSID) à Node-red. Cette transmission se fait via une requête HTTP POST, comme vu précédemment.
- **Traitement des données dans Node-red** : Node-red reçoit ces données et les traite. Les données sont enregistrées dans une variable globale dans la fonction 1 (via la fonction JavaScript flow.set(« Nom variable », données). Par la suite, elles sont enregistrées dans le msg.payload dans la fonction 2. Cela permet ensuite d'envoyer ces données dans la charge utile de la requête GET.

- **Envoi de la requête GET par Node-red :** Une fois les données traitées, Node-red envoie une requête GET au serveur Flask. Cette requête inclut les données Wi-Fi formatées par le code Arduino dans l'étape 1, au format JSON.
- **Réception et validation des données par le Serveur Flask :** Le serveur Flask, à la réception de la requête GET, extrait les données Wi-Fi du corps de la requête. Le serveur valide ensuite ces données pour s'assurer qu'elles sont dans un format correct et utilisables.

Calcul de la position de l'ESP32 basée sur les Données Wi-Fi

Notre enseignant nous ayant fourni un fichier CSV contenant les informations de position détaillées des réseaux WiFi captés par son smartphone, nous avons donc pu procéder au calcul de la position de notre appareil en utilisant ces données.

Après la réception des données Wi-Fi par le serveur Flask via une requête GET, le calcul de la position de l'ESP32 est effectué comme suit :

- **Base de Données Wi-Fi** : Comme dit précédemment, notre enseignant nous a fourni un fichier CSV contenant les informations de localisation des réseaux Wi-Fi captés par son appareil. Nous avons utilisé ce fichier pour créer une base de données SQLite avec un script Python. Ce script a créé la table wifi_networks.db et chargé les données du CSV, comprenant SSID, RSSI, BSSID, latitude et longitude, ainsi que d'autres informations que nous n'avons pas utilisées.
- **Traitement des Données Wi-Fi :** Comme décrit dans la partie précédente, le serveur Flask reçoit les données Wi-Fi (SSIDn BSSID et RSSI) au format JSON, et les utilise pour déterminer la position de l'ESP32, dans la fonction locate_device(wifi_networks), wifi_networks étant les données des réseaux WiFi envoyées par l'ESP32, au format JSON.
- **Calcul de la Position la Plus Proche** : La fonction get_closest_location parcourt les informations sur les réseaux reçus et cherche l'emplacement avec le RSSI le plus proche de celui reçu, pour chaque BSSID (dans une boucle *for* parcourant l'ensemble des réseaux). -> appel à get_rssi_weighted_positions(bssid, rssi).
- **Calcul des Positions Pondérées par RSSI :** Par la suite, la fonction get_rssi_weighted_positions calcule une position pondérée pour chaque BSSID en fonction de la différence de RSSI, offrant ainsi une estimation plus précise de la position. Cela nous a demandé de requêter la base de données, mais les données étant écrites en minuscule, et nos données reçues par l'ESP32 contenant des caractères majuscules, nous avons alors connu une erreur qui nous a demandé de convertir nos chaines de caractères en minuscule.
  Voici le détail du fonctionnement de cette fonction, qui est probablement la plus compliquée de notre programme :

1. **Interrogation de la Base de Données** : La fonction commence par interroger la base de données SQLite que nous avons créé à partir du fichier CSV fourni, pour trouver toutes les entrées correspondant au BSSID donné. Pour chaque entrée, elle récupère la latitude, la longitude et le RSSI.
2. **Calcul des Poids Basés sur le RSSI :** Pour chaque emplacement trouvé dans la base de données, l'algorithme calcule un poids. Ce poids est déterminé par la différence entre le RSSI stocké dans la base de données et le RSSI reçu de l'ESP32. Plus cette différence est petite, plus le poids est élevé, indiquant que l'emplacement de la base de données est plus proche de la position actuelle de l'ESP32.
3. **Application des Poids aux Coordonnées** : Chaque couple de latitude et longitude est ensuite pondéré par le poids calculé. Cela signifie que chaque position est ajustée en fonction de sa probabilité d'être proche de la position réelle de l'ESP32.
4. **Compilation des Positions Pondérées :** Toutes ces positions pondérées sont regroupées pour être utilisées dans le calcul de la position moyenne pondérée.

- **Calcul de la Position Moyenne Pondérée :** locate_device compile toutes ces positions pondérées dans un tableau pour estimer la position actuelle de l'ESP32 :

    - **Calcul du Poids Total** : Les poids de toutes les positions pondérées sont additionnés afin d'obtenir le poids total. Ce total représente la somme des influences de chaque position en fonction de leur RSSI respectif.
    - **Vérification du Poids Total** : Si le poids total est supérieur à zéro, cela signifie que des positions valides ont été trouvées et peuvent être utilisées pour le calcul. Si le poids total est nul, cela indique qu'aucune position valide n'a été trouvée. Dans ce cas une position nulle sera retournée.
    - **Calcul des Coordonnées Moyennes Pondérées** : Si le poids total est positif, l'algorithme calcule la latitude et la longitude moyennes. Cela se fait en multipliant chaque latitude et longitude par son poids, en les additionnant, puis en divisant le total par le poids total. Cela donne la position moyenne pondérée, qui est une estimation de la position de l'ESP32.
    - **Retour de la Position** : Enfin, l'algorithme renvoie les coordonnées moyennes pondérées calculées. Si aucune position valide n'est trouvée, il renvoie un ensemble de coordonnées par défaut (0,0).

- **Renvoi de la Position Calculée :** Enfin, le serveur Flask renvoie les coordonnées calculées en réponse à la requête GET. Le serveur Node-red les reçoit donc et cela permet ensuite d'afficher la position sur une carte.

Affichage de la position sur une carte

Après avoir calculé la position de l'ESP32, la dernière étape est son affichage sur une carte :

- **Réception de la Position par Node-red :** Le serveur Flask renvoie les coordonnées calculées en réponse à la requête GET. Node-red, ayant initié la requête, reçoit ces coordonnées.
- **Traitement dans Node-red:** Node-red traite ces données pour les préparer à l'affichage. Le nœud « *worldmap* » de node-red nécessite une entrée au format JSON, elle est donc formattée dans la réponse du serveur Flask :

```
formatted_data = {
        "name": "Device",
        "lat": current_latitude,
        "lon": current_longitude,
}
```

Nous utilisons ensuite un nœud « *json* » afin de convertir cette chaine en objet JavaScript. (cf figure 1)

- **Affichage sur la Carte :** Les coordonnées sont par la suite envoyées au nœud « *worldmap* ». Cela permet de visualiser la position estimée de l'ESP32 sur une carte interactive, avec précision.
- **Mise à Jour Dynamique :** Si de nouvelles données de position sont reçues, Node-red met à jour l'affichage sur la carte automatiquement, permettant ainsi un suivi dynamique de la position de notre ESP32.

Conclusion

Ce projet de géolocalisation sans GPS nous a permis de mettre en pratique nos compétences en programmation Python et Arduino, ainsi que la mise en œuvre d'un serveur node-red.

En utilisant l'ESP32 pour scanner les réseaux Wi-Fi et en traitant ces données pour calculer une position, nous avons démontré qu'il est possible de localiser un appareil avec une précision raisonnable sans utiliser le GPS (sous réserve d'avoir à disposition une base de données contenant les localisations des réseaux WiFi). La création d'une base de données à partir d'un fichier CSV et l'utilisation de Node-red et Flask pour gérer les données et afficher la position sur une carte ont également été des aspects très intéressants.