Robotique et Domotique Communicantes
TP4/TP5 : Smart Building

Clara Schild, Lucas Schaeffer, Thomas Rives, Yassine Lambarki

11 Janvier 2021

Table des matières

1 Introduction

Dans ce TP, nous allons étudier deux méthodes d'envoi de données. La première méthode consiste à utiliser la norme IEEE 802.15.4 pour envoyer les données à un service Cloud (Cayenne). La seconde consiste à utiliser le serveur LoRaWAN pour faire remonter les données. Dans les deux cas, nous utiliserons 10 Zigduinos capteurs (Device) et un Zigduino passerelle. Nous étudierons ensuite les différents avantages et inconvénients de ces deux méthodes, avant d'en déduire leurs cas d'utilisation.

2 Catégories de réseaux

On recense deux grandes catégories de réseaux, que l'on va présenter ci-dessous.

2.1 Les réseaux à courte portée

Les réseaux à courte portée permettent de transférer des données sur de faibles distances (inférieures à 100 m). Ils sont très utilisés dans la domotique, notamment au sein d'une maison, ou pour des objets connectés pouvant être portés. Les bracelets connectés, par exemple, ne sont jamais très loin du téléphone auquel il transfère ses données.

2.2 Les réseaux de longue portée et basse consommation

Les réseaux de longue portée et basse consommation (`LPWAN`) permettent de transférer des données d'un appareil à l'autre sur de vastes distances (de quelques centaines de mètres à plusieurs dizaines de kilomètres). Ils sont utilisés par des entreprises qui veulent connecter des kilomètres d'infrastructures à Internet ou dans des projets de Smart Cities.

Les `LPWAN` utilisent les bandes de fréquences à usage libre sans licence appelées `ISM`. Leur utilisation implique le partage des ressources avec les concurrents (SigFox) et avec les autres technologies (Wi-Fi, Zigbee, Bluetooth et d'autres).

3 Lecture des données

Pour se familiariser avec l'environnement, nous avons tout d'abord essayé d'afficher un simple message sur le terminal distant. Nous avons initialisé la liaison série via `Serial.begin(57600)`, pour pouvoir récupérer les caractères ASCII avec la commande `nc`.

Exécuter `nc nom_noeud 20000` nous permet d'interagir avec un noeud sur le port 20000.
On visualise ainsi toutes les données qu'un Zigduino feuille peut envoyer à la passerelle.

Les mesures des capteurs sont systématiquement envoyées vers le Zigduino `sink`. Pour renseigner l'identité de ce `sink`, on modifie la ligne :
`if (linkaddr_node_addr.u8[6] == 0 && linkaddr_node_addr.u8[7] ==` XX), où XX représente le numéro du Zigduino auquel nous souhaitons attribuer le rôle de `sink`.

Le `sink` centralisera donc toutes les données. Elles seront ensuite envoyées à un script Python qui tournera sur le serveur `IoT-LAB`, où ce dernier s'occupera de leur traitement. Pour récupérer les données, ce script se connecte au port série du Zigduino racine à l'aide d'une socket (sur le port 20000).

Si la ligne est détectée comme reçue de la part d'un end-device (grâce à la présence de `sink_received` dans la ligne), on extrait les données suivantes :
- le numéro du Zigduino (pour identifier la provenance des données).
- la température ambiante.
- la luminosité ambiante.
- l'humidité ambiante.
- le volume sonore.
- la présence d'une entité.

Pour envoyer ces données au Cloud, on utilisera 2 supports différents qui seront détaillés dans la suite.

4 Déploiement d'un réseau courte-portée IEEE 802.15.4

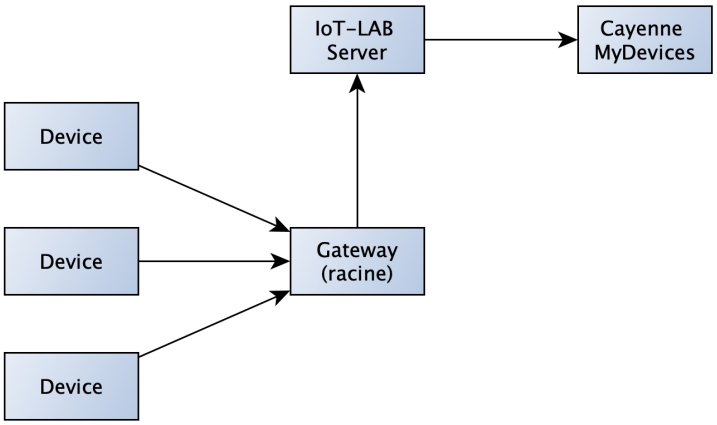

FIGURE 1: Topologie du réseau courte-portée IEEE 802.15.4

Pour cette première partie, nous avons utilisé `Rime Collect` pour récupérer les données de dix Zigduino sur notre Zigduino passerelle, aussi appelé `sink` (le 86 : `zigduino-86.strasbourg.iot-lab.info`).

Comme on peut le voir sur la figure 1, la topologie en étoile est utilisée pour ce réseau. La passerelle, le sink, correspond à la racine du réseau et collecte les données des autres Zigduino. Les Devices sont des noeuds feuilles et envoient périodiquement les valeurs de leurs capteurs à la passerelle.

Nous avons utilisé l'OS `Contiki` avec l'application `Rime Collect`. Notre réseau de collecte utilise le protocole `IEEE 802.15.4`. Pour éviter les interférences, nous utilisons le canal radio 13.

Pour que les Zigduinos puissent identifier le `sink`, on modifie la ligne :
`if(linkaddr_node_addr.u8[6] == 0 && linkaddr_node_addr.u8[7] == 86)`.
Elle indique à présent que le Zigduino 86 est le `sink`, et que les données doivent lui être envoyées.

Après avoir compilé le code de `Rime Collect`, nous pouvons mettre à jour le firmware sur tous les Zigduinos : `iotlab-node --update collect-sensors.avr-zigduino`.

4.1 Transfert des données vers le Cloud Cayenne MyDevices

Le script Python expliqué dans la partie 3, isole les données transmises et les envoient à Cayenne, où elles seront affichées sur un tableau de bord.

Dans ce script, si le numéro du Zigduino est différent de celui du `sink`, on envoie les données vers le Cloud (via les fonctions `celsiusWrite`, `luxWrite` et `virtualWrite`) dans le canal dédié à celui-ci, ce qui actualise les valeurs affichées par le tableau de bord.

Dans un second temps, on se connecte au Cloud Cayenne de MyDevices. Sur le tableau de bord IoT-Zigduino, on observe les cinq différents types de données pour chaque Zigduino.

FIGURE 2: Tableau de bord sur Cayenne MyDevices pour le réseau courte portée

Nous n'observons pas de délai entre une réception de données de la part des capteurs et l'émission de données vers le Cloud Cayenne. Les données envoyées par un Zigduino feuille actualisent les valeurs observées sur Cayenne sans attendre. Par contre, il faut prendre en compte le temps de transmission qui est relativement considérable.

5 Déploiement d'un réseau longue portée LoRaWan

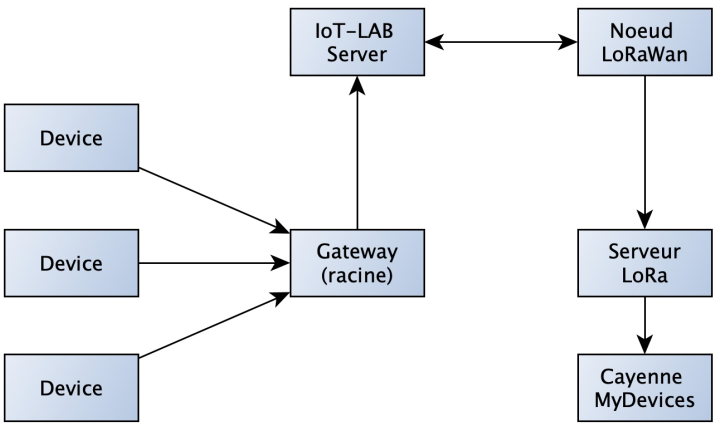

FIGURE 3: Topologie du réseau longue portée LoRaWan

Dans cette seconde partie, on utilise toujours `Rime Collect`. Cependant, nous avons changé de `sink`. C'est le Zigduino 14 (`zigduino -14. strasbourg.iot -lab.info`) qui occupe désormais ce rôle.

La figure ci-dessus illustre la topologie utilisée pour ce réseau. Les Devices envoient les données à notre Gateway (le Zigduino 14, le noeud racine). Le script Python permet de récupérer et de parser ces données avant de les envoyer au noeud LoRaWAN. C'est ce dernier qui va communiquer les données au serveur LoRa et ensuite à Cayenne MyDevices.

La ligne `if(linkaddr_node_addr.u8[6] == 0 && linkaddr_node_addr.u8[7] == 86)` a de nouveau été modifiée pour devenir `if(linkaddr_node_addr.u8[6] == 0 && linkaddr_node_addr.u8[7] == 14)` afin de changer l'identité du `sink`.

Dans cette partie, on utilise deux firmwares différents. Les Zigduinos capteurs ainsi que la Gateway conservent le même firmware que dans la partie précédente. En revanche, le Zigduino 14 se voit attribuer un firmware différent. Ce dernier a été compilé grâce à l'environnement Arduino. Le fichier d'extension `elf` a été flashé uniquement sur ce Zigduino. Son rôle particulier explique la nécessité de lui affecter un firmware spécial.

Pour cette partie, nous avons utilisé 3 types de Zigduinos :
- 10 Zigduinos capteurs.
- 1 Zigduino collecteur qui a le rôle de racine du réseau 802.15.4 (qui récupère les données mesurées parmi les Zigduinos capteurs et les envoient à un script Python).
- 1 Zigduino LoRaWan (qui lit des données depuis son port série pour les envoyer au LoRaServer en utilisant LoRaWAN).

5.1 Interface entre la racine du réseau 802.15.4 et le noeud LoRaWan

Comme précisé précédemment, nous utilisons deux noeuds différents : un pour la collecte des données (le Zigduino Gateway, racine, sous `Contiki`) et un pour envoyer les données au serveur LoRa (sous Arduino).

Pour faire le lien entre ces deux noeuds, un script Python tourne sur le serveur IoT-LAB. Celui-ci récupère les données par le Zigduino racine et les envoient au noeud LoRaWan. Ainsi, deux sockets sont utilisées pour se connecter sur le port série des deux Zigduinos.

On profite de cette étape de transition pour parser les données reçues par le Zigduino racine avant de les envoyer au noeud LoRaWAN.

Les données parsées et envoyées au noeud LoRa sont de la forme suivante :
`no_zigduino temp hum sound light pir`.

Pour synchroniser l'envoi des données au noeud LoRaWAN, nous utilisons des acquittements. Ainsi, le script Python attend de recevoir l'acquittement avant d'envoyer la prochaine donnée.
Nous avons fait en sorte de ne perdre aucune donnée. La synchronisation permet de ne pas attendre inutilement avant de pouvoir renvoyer des données.

5.2 Connexion au serveur LoRaWan

Le noeud LoRaWan va nous permettre de nous connecter sur le serveur, qui permet l'intégration de Cayenne pour relayer les messages reçus au Cloud. Sur ce noeud, nous faisons tourner un code Arduino. Pour ce faire, nous avons repris le code mis à disposition pour générer une DevEUI personnalisée et nous avons ajouté des fonctions pour récupérer les données des Zigduinos capteurs. Ces données sont transmises par le script Python qui tourne sur le serveur IoT-LAB.

Dans le `setup()`, on définit la DevEUI personnalisée et les clés pour se connecter au serveur LoRa via une connexion OTAA avec `joinOtaa(appEui, appKey)`.

Dans le `loop()`, on attend que le port série soit disponible et on récupère le nombre de données qui vont être transmises. Ensuite, on décode les données reçues et on utilise le codec Cayenne pour encoder le message à transmettre au serveur LoRa. On précise également le canal sur lequel devra s'afficher la donnée sur Cayenne. Puis, on envoie le message grâce à la librairie `TheThingsNetwork` et la méthode `sendBytes(buffer, size)`.

Après que le message soit envoyé à Cayenne MyDevices, un acquittement est transmis au script Python pour lui indiquer qu'il peut envoyer la donnée suivante.

5.3 Redirection des données vers le Cloud Cayenne MyDevices

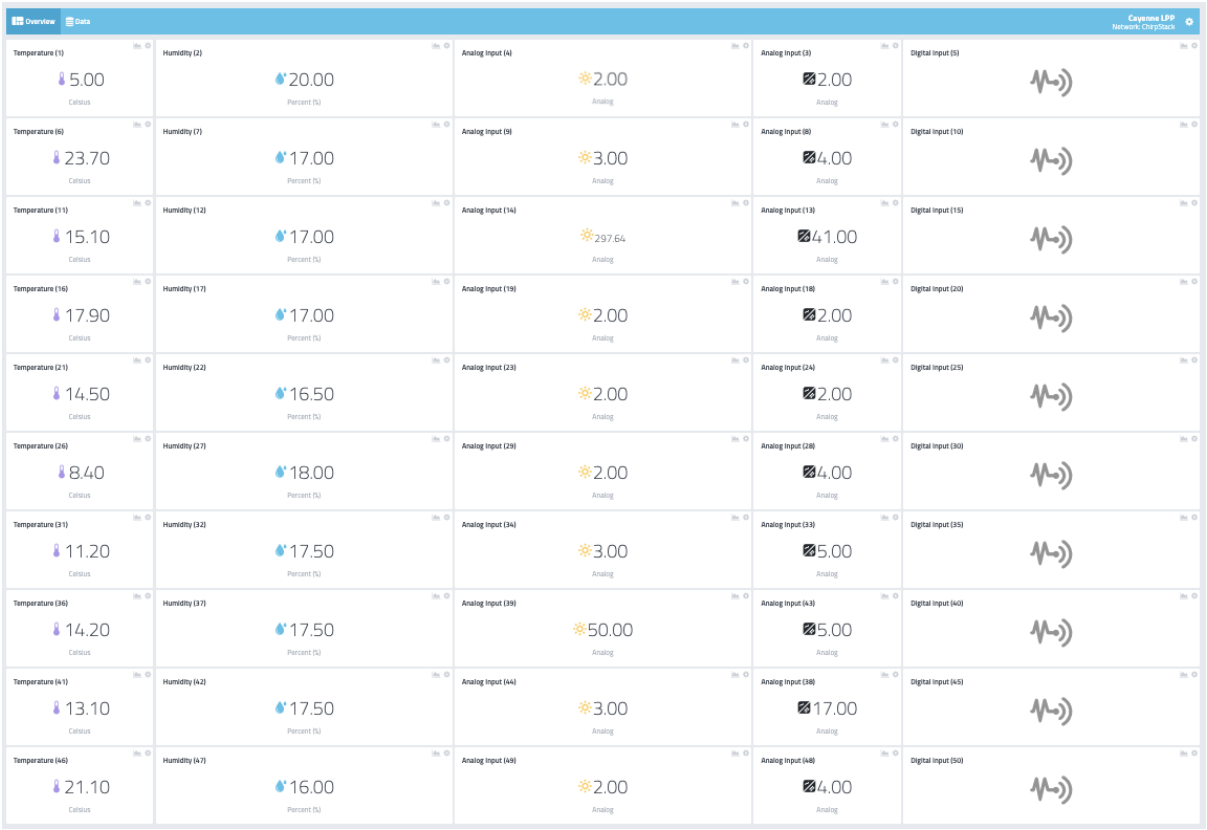

FIGURE 4: Tableau de bord sur Cayenne MyDevices pour le réseau longue portée

6 Analyse des résultats

	LoRaWan	IEEE 802.15.4
Portée	Longue portée - zones rurales : entre 15 et 20 km - zones urbaines : entre 3 et 8 km	Portée <100 m
Débits	250 à 5470 bits/s	256 Kbits/s
Qualités des produits	Appareils souvent peu chers et de moindre qualité	Coût moyen, qualité plus élevée
Topologie réseaux	Réseau en étoile avec noeud connecté à la passerelle Simple à utiliser mais nécessite d'être a portée de la passerelle, use moins rapidement les batteries, ce qui permet d'allonger la durée de vies des appareils	Réseau en maille, flexible, fiable et extensible. Communication peer-to-peer pour des liaisons directes à grande vitesse ou alors d'un noeud à la passerelle
Consommation	Basse consommation. La transmission des données est favorisée. Permet aux objets connectés d'échanger des données généralement de faible taille.	Consommatrice en batterie
Fiabilité du réseau	Perte de paquets qui peut s'élever de 10% à 75%. Il n'y a pas d'acquittements intégrés, c'est à l'ingénieur réseau de concevoir un système fiable. La topologie en étoile ne permet pas de la communication peer-to-peer. Étant donné que `LoRaWan` opère sur un large spectre de fréquences, le risque d'interférences est potentiellement grand. Ce qui fait que les paquets sont souvent perdus. Sur 1000 messages par minute, 45% d'entre-eux sont perdus à cause de collisions. Même si on diminue le nombre de messages envoyés aux gateaway à 100, 10 % n'arrivent pas	Fiabilité élevée avec un système d'accusé de réception intégré : tout paquet perdu est renvoyé automatiquement. Les derniers transmetteurs Zigbee ont des architectures de réseau maillé auto-réparatrice, c'est-à-dire que si un noeud du réseau tombe, les autres noeuds seront automatiquement rédigés
Durée de vie de la batterie	Dépend du fabricant et des batteries utilisées mais généralement une très longue durée de vie. 1,5 an et plus pour les nœuds endormis selon l'appareil	Dépend du fabricant et des batteries utilisées mais généralement une très longue durée de vie. 1,5 an et plus pour les nœuds endormis selon l'appareil
Taux d'interrogation	Généralement des taux d'interrogation plutôt lents	Interrogation continue en temps réel à grande vitesse, une fréquence possible de 50ms jusqu'à 10 secondes, avec des cycles de sommeil plus longs
Applications	Utilisé pour des applications de récupération d'information de faible puissance	Pour des applications de récupération d'information et d'automatisation, y compris la surveillance et le contrôle en temps réel
Domaine d'applications	Agriculture, Smart Cities, environnement	Industrie, bâtiment, construction, environnement, agriculture, Smart Cities, énergie

Table 1: Comparaison entre LoRaWan et IEEE 802.15.4

La norme IEEE 802.15.4, plus économe en énergie, est envisageable à l'échelle du bâtiment. Elle permet de créer un réseau maillé mais la configuration reste complexe à mettre en oeuvre à l'échelle d'une ville. Les technologies radio longue portée comme LoRaWan permettent ce passage à l'échelle avec des débits suffisants pour effectuer des actions à distance.

Pour garantir une couverture radio totale, l'utilisateur devrait déployer son propre réseau d'antennes pour couvrir les éventuelles zones non couvertes. Dans ce cas de figure, LoRaWan est plus simple à mettre en place.

LoRaWan convient aux applications qui ont besoin de communications longue portée et qui utilisent des objets de faible consommation.

Si l'application n'a pas d'exigence de temps réel, de contrôle ou d'automatisation et que des taux d'interrogation plus lents conviennent, alors LoRaWan est un bon choix.

IEEE 802.15.4, d'un autre côté, est idéal pour les applications qui nécessitent une meilleure fiabilité, surveillance en temps réel, contrôle ou automatisation.

7 Remarques et difficultés

Nous avons eu beaucoup de mal à réaliser ce TP. Pour la première partie, nous avons essayé d'utiliser CoAP, sans succès. L'utilisation de Rime Collect nous a beaucoup aidé. L'écriture du script Python ne présentait pas de difficultés majeures. Cependant, dans un premier temps, nous n'avions pas pensé à exécuter un script sur le serveur IoT-LAB.

La partie 5 a posé beaucoup plus de problèmes. Nous étions complètement perdus et ne savions plus quoi faire. Nous avons testé de nombreuses approches, de nombreux codes différents, et de nombreuses méthodes sans que celles-ci ne portent leur fruit.