Université de Strasbourg

UFR Maths-Info/TPS

M2SIRIS/RIO3A

Année 2023/2024

IoT

TP 1 : LPWAN

Introduction

Ce TP a pour but de configurer des objets connectés longue portée (LPWAN) pour les cas d'usages liés à la Smart City. Nous utiliserons pour cela deux services d'expérimentation :

- le service IoT-LAB pour réserver et reprogrammer un objet End-Device LoRaWAN
- le service LRP IoT de la plateforme ICube Inetlab pour enregistrer un objet End-Device sur infrastructure LoRaWAN

Les informations de connexion sont dans le fichier « Logins LRP IoT + IoT-LAB » présent sur Moodle.

1 Réservation de l'objet sur IoT-LAB

1. A l'aide de son compte IoT-LAB, se connecter sur l'interface web IoT-LAB

https://www.iot-lab.info/

2. Ajouter une clé SSH présente sur votre poste de travail en suivant la documentation suivante :

https://www.iot-lab.info/docs/getting-started/ssh-access/

3. Depuis l'interface web, cliquer sur l'icône « + » bleue (New Experiment) et programmer une nouvelle expérimentation avec un seul objet Pycom sur le site de Strasbourg :
    - https://www.iot-lab.info/testbed/experiment
    - name : tp1_iot_stras
    - duration : 240 minutes
    - Start : ASAP
    - Nodes : Select by Node Properties
        - Architecture : **Pycom (LoPy)** ou **Pycom (Fipy)**
        - Site : **Strasbourg** (obligatoire pour accrocher le réseau LoRaWAN de Strasbourg)
        - Qty : **1**

- Cliquer sur **Submit Experiment**

2 LoRaWAN – class A

A l'aide de son compte LRP IoT, se connecter sur l'interface web du serveur LoRaWAN :

- https://inetlab-lorawan.icube.unistra.fr

L'implémentation utilisée est le projet open source ChirpStack.

1. Créer un profil d'objet (*Device-Profile*) de classe A en mode OTAA :

https://inetlab.icube.unistra.fr/index.php/Documentation/Doc-LRP_IoT-organisation#Comment_cr.C3.A9er_un_profil_d.27acc.C3.A8s_r.C3.A9seau_.3F_.28Device-profiles.29

General >

Device-profile name : **class-A-otaa-1.0.2**

LoRaWAN MAC version : **1.0.2**

LoRaWAN Regional Parameters revision : **A**

ADR algorithm : **Default**

Flush queue on activate: **True**

Expected uplink interval (secs) (seconds) : **3600**

Device-status request frequency (req/day) **: 1**

Join (OTAA/ABP) >

Device supports OTAA **: True**

Class-B >

Device supports Class-B **: False**

Class-C >

Device supports Class-C **: False**

Codec >

Payload codec **: Cayenne LPP**

Relay >

Device is a Relay **: False**

Device is a Relay capable end-device **: False**

2. Créer une application (*Applications*) intitulée **tpiot** :

https://inetlab.icube.unistra.fr/index.php/Documentation/Doc-LRP_IoT-organisation#Comment_cr.C3.A9er_une_application_.3F_.28Applications.29s

3. Déclarer un objet (*End-Device*) dans l'application **tpiot** :

Attention, pour les champs *Device EUI (EUI64)*, il faudra générer une **adresse aléatoire** (cliquer sur la flèche *Generate Random ID*) :

https://inetlab.icube.unistra.fr/index.php/Documentation/Doc-LRP_IoT-organisation#Comment_ajouter_des_End-Devices_.C3.A0_une_application_.3F

3 Programmation de l'objet

La procédure d'utilisation de l'IDE à distance sur un objet de la plateforme IoT-LAB est validée sous un environnement de travail **GNU/Linux** pour pouvoir rapidement flasher l'objet depuis l'IDE (Atom ou Visual Studio). **Note** : cette procédure ne fonctionne pas nativement avec Pymakr sous MacOSX (solution possible rshell, mais moins pratique à l'usage).

1. Installer l'IDE **VSCode** (v1.82.2) et le plugin **Pymakr** (Pymakr - Preview`v2.25.2`) comme décrit sur le site de Pycom :

https://docs.pycom.io/gettingstarted/software/vscode/

2. Installer l'utilitaire Linux **socat** qui permet d'encapsuler/décapsuler les flux d'un port série sur une socket TCP:

apt-get install socat nodejs npm

3. Ouvrir un **premier terminal** et taper la commande suivante en remplaçant le numéro <id> de l'objet pycom réservé et votre login IoT-LAB:

ssh -L 20000:**pycom-<id>**:20000 <iotlab_login>@strasbourg.iot-lab.info

4. Ouvrir un **deuxième terminal** et taper la commande suivante:

socat -d -d pty,link=/tmp/ttyS0,echo=0,crnl TCP:127.0.0.1:20000s

5. Dans un troisième terminal cloner le dépôt git suivant :

git clone https://github.com/schrein/lopy-example

6. Lancer VSCode et ouvrir le projet **eb_demo** du dépôt lopy-example
7. Paramétrer le plugin Pymakr pour utiliser le port série **/tmp/ttyS0** créé avec socat.
8. Modifier le fichier **config.py** du projet pour paramétrer **APP_KEY** et **DEV_EUI** correspondant à l'objet précédemment déclaré sur le serveur LoRaWAN.
9. Uploader le code sur l'objet et vérifier la bonne réception des messages sur le serveur.

NOTE : ne pas hésiter à utiliser la commande reset de l'objet via l'interface IoT-LAB. L'upload du code MicroPython via VSCode et Pymakr ne redémarre pas automatiquement l'objet avec le nouveau code.

10. Envoyer un Downlink **sans** acquittement via l'interface ChirpStack avec le chaine en base64 suivante : **dGl0aXRvdG8=** . Observer les échanges de messages dans l'onglet LoRaWAN Frames. Vérifier la réception du message Downlink sur l'objet.
11. Envoyer un Downlink **avec** acquittement via l'interface ChirpStack avec le chaine en base64 suivante : **dGl0aXRvdG8=** . Observer les échanges de messages dans l'onglet LoRaWAN Frames. Vérifier la réception du message Downlink sur l'objet. Comment le message downlink a bien été acquitté ?

4 LoRaWAN – class C

Reprendre l'exercice 2 pour déclarer un objet de classe C :

1. Faire un nouveau *Device-Profile* intitulé, **class-C-otaa-1.0.2**
2. Déclarer un nouveau End-Device avec le *Device-Profile* **class-C-otaa-1.0.2** dans la même *Application* **tpiot**, en générant un nouvel DevEui et une nouvelle AppKey.

Repartir de l'exercice 3 pour coder un objet de classe C :

3. Après la phase d'enregistrement sur le réseau (Join), votre objet doit envoyer un message Uplink. Puis il se met en attente de réception de plusieurs message Downlink de la part du serveur. A la réception d'un Downlink, votre objet doit renvoyer dans la foulée un Uplink.
4. Envoyer un Downlink **sans** acquittement via l'interface ChirpStack avec le chaine en base64 suivante : **dGl0aXRvdG8=** . Observer les échanges de messages dans l'onglet *LoRaWAN Frames*. Vérifier la réception du message Downlink sur l'objet.
5. Envoyer un Downlink **avec** acquittement via l'interface ChirpStack avec le chaine en base64 suivante : **dGl0aXRvdG8=** . Observer les échanges de messages dans l'onglet *LoRaWAN Frames*. Vérifier la réception du message Downlink sur l'objet. Comment le message Downlink a bien été acquitté ?

5 LoRa Raw messages

Dans cet exercice, nous allons travailler en binôme avec deux Pycoms qui vont émettre et recevoir des messages directement en point à point en utilisant la modulation LoRa sans passer par l'infrastructure LoRaWAN. Pour cela, vous pouvez vous inspirer des exemples suivants :

https://docs.pycom.io/tutorials/networks/lora/lora-mac/

https://docs.pycom.io/tutorials/networks/lora/module-module/

1. Proposer des améliorations pour limiter les interférences au niveau MAC.
2. Filtrer les messages reçus au niveau applicatif.
3. Vérifier si le « 1 % max duty cycle » LoRa est implémenté également avec les messages en mode LoRa Raw ou si cette limite est à la discrétion du développeur.

https://avbentem.github.io/airtime-calculator/ttn/eu868

6 Références

https://www.iot-lab.info/docs/getting-started/introduction/

https://inetlab.icube.unistra.fr/index.php/Documentation/Doc-LRP_IoT-organisation

https://www.chirpstack.io/docs/chirpstack/use/index.html

https://www.iot-lab.info/docs/boards/pycom/

https://docs.pycom.io

https://micropython.fr/05.outils/terminal_serie/rshell/

https://github.com/dhylands/rshell