Rapport du projet

Terminaux Mobiles Communicants

IoT, LoRa, WiFi, MQTT, SSL, ATECC508, Mongoose OS, Raspberry Pi ESP8266

Réalisé par :

Fatima Ez-zahrae Elfara
Florian SUCHAUD

Encadré par :

P-F. Bonnefoi

Table des figures

1.1 Architecture de projet 2

Table des matières

1 Configuration de l'architecture Installation des Outils **1**
Introduction 1
1.1 Installation de l'host 2
1.2 Installation de dnsmasq et hostapd 4
1.2.1 Dnsmasq 4
1.2.2 Hostapd 4

2 Communication et Sécurité **5**
2.1 Mise en place des communications LoRa 5
2.2 Mise en place du serveur mosquitto 5
2.3 Mise en place de l'ESP 7

3 Déploiement de projet **9**

4 Démonstration **10**

Chapitre 1

Configuration de l'architecture Installation des Outils

Introduction

Nous avons pour objectif de réaliser un projet visant à mettre en place un réseau de capteurs connectés par WiFi à un concentrateur. Le but de ce réseau de capteurs est de remonter des mesures vers le concentrateur en utilisant le protocole MQTT, permettant ainsi de collecter des données de manière efficace et fiable.

Chaque capteur sera équipé d'un ESP8266 intégré dans une carte de développement Wemos, tandis qu'un Raspberry Pi jouera le rôle de concentrateur en exécutant un broker MQTT (logiciel mosquitto) et en servant de point d'accès WiFi (logiciel hostapd).

Pour assurer l'authentification du serveur MQTT lors de la connexion en TLS et l'authentification du client auprès du serveur MQTT, chaque capteur exploitera un circuit dédié à la manipulation de la cryptographie sur courbe elliptique ATECC608, connecté à l'ESP8266 par le bus I2C.

Nous utiliserons le framework de développement Mongoose OS pour programmer le système embarqué ESP8266, disposer d'une implémentation de TLS et exploiter le composant ATECC608 pour réaliser les opérations de chiffrement, signature et vérification.

Enfin, le concentrateur sera relié à une passerelle en utilisant les communications LoRa. Chaque Raspberry Pi sera équipé d'un dragino intégrant un transceiver LoRa et un GPS, permettant ainsi la communication des mesures vers le Raspberry Pi connecté à Internet.

1.1 Installation de l'host

Dans ce projet d' IoT. On a Deux raspberrys communiquent entre eux par LoRa, communication chiffrée par AES. Les fichiers des rapberry sont hébergés et utilisés depuis un ordinateur hote par connection ethernet.
Voici l'Architecture de projet :

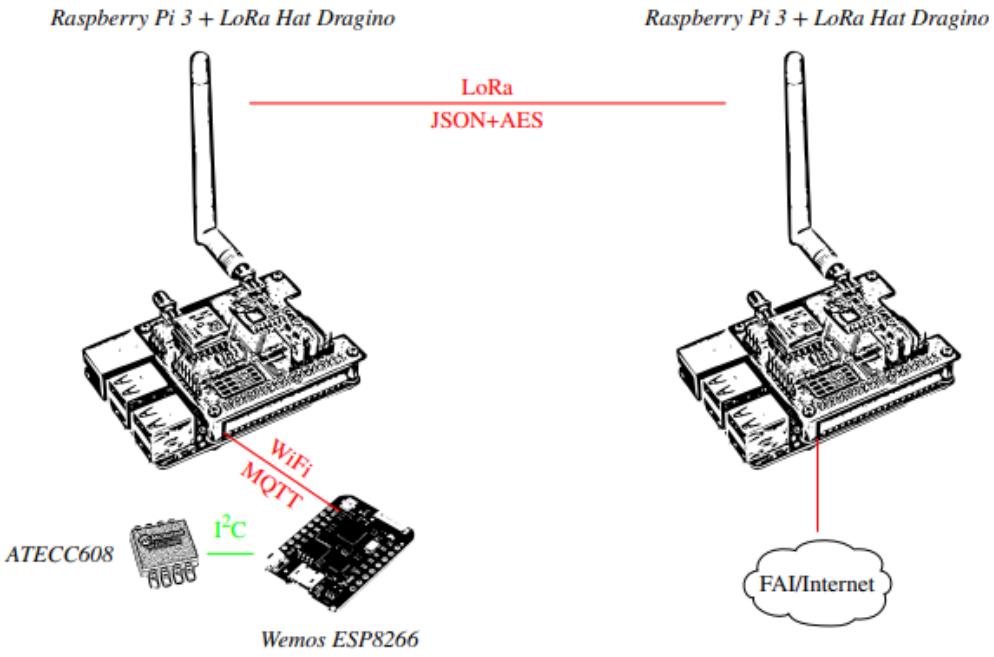


FIGURE 1.1 – Architecture de projet

— Afin de commencer l'installation de l'host nous allons créer notre répertoire ainsi que télécharger, et monter l'OS de notre raspberry. Ici notre path sera /home/florian/TMCProject/RASPI, il faut donc l'adapter à notre situation.

```
$ mkdir RASPI
$ cd RASPI
$ mkdir client
$ mkdir boot
$ wget https://downloads.raspberrypi.org/raspios_lite_arm64/
   images/raspios_lite_arm64-2022-09-26/2022-09-22-raspios-
   bullseye-arm64-lite.img.xz
$ unxz 2022-09-22-raspios-bullseye-arm64-lite.img.xz
$ sudo losetup -fP 2022-09-22-raspios-bullseye-arm64-lite.
   img
$ losetup -a | grep rasp
```

— Suite à cette commande on récupère le numéro du loop correspondant à notre image. Pour les commandes suivantes ce numéro sera remplacé par un X.

```
$ sudo mount /dev/loopXp2 /mnt
$ sudo rsync -xa --progress /mnt/ client/
$ sudo umount /mnt
$ sudo mount /dev/loopXp1 /mnt
$ cp -r /mnt/* boot/
$ sudo umount /mnt
```

— Nos systèmes de fichiers sont désormais presque prêt pour démarrer notre raspberry, il reste cependant la configuration à faire. Pour ce faire nous allons devoir activer le NFS.

```
$ sudo apt install nfs-kernel-server
```

— On modifie également le fichier /etc/exports en ajoutant à la fin les lignes :

```
/home/florian/TMCProject/RASPI/client *(rw,sync,
   no_subtree_check,no_root_squash)
/home/florian/TMCProject/RASPI/boot *(rw,sync,
   no_subtree_check,no_root_squash
```

— Puis on active et relance nos services.

```
$ sudo systemctl enable nfs-kernel-server
$ sudo systemctl enable rpcbind
$ sudo systemctl restart nfs-kernel-server
```

— On récupère ensuite le nom de notre interface réseau Ethernet grâce à la commande :

```
$ ip a
```

— On remplacera donc le nom de l'interface par celle-ci dans le script "script_boot_rpi". On pensera aussi à y modifier le path dans la dernière ligne du script.On modifie aussi le fichier cmdline.txt :

```
console=serial0,115200 console=tty1 root=/dev/nfs nfsroot
   =10.20.30.1:/home/florian/TMCProject/RASPI/client,tcp,
   vers=3 rw ip=dhcp rootwait
```

— Puis le fichier /client/etc/fstab :

```
proc /proc proc defaults 0 0
10.20.30.1:/home/florian/TMCProject/RASPI/boot /boot nfs
   defaults,vers=3 0 0
```

— Puis on édit le fichier /client/lib/systemd/system/sshswitch.service afin qu'il ressemble à ça :

```
[Unit]
Description=Turn on SSH if /boot/ssh is present
After=regenerate_ssh_host_keys.service
[Service]
Type=oneshot
ExecStart=/bin/sh -c "systemctl enable --now ssh"
[Install]
WantedBy=multi-user.target
```

Il ne reste donc plu-qu'à démarrer le script et nous connecter à notre raspberry.

1.2 Installation de dnsmasq et hostapd

1.2.1 Dnsmasq

Dnsmasq est un package qui permet d'instancier un serveur DNS et DHCP au sein de la configuration Raspberry Pi. Il facilitera l'utilisation de l'ESP8266.
L'installation se fait avec la commande suivante :

```
apt-get install dnsmasq
```

Par la suite, nous avons configuré l'interface de DNS,les adresses à utiliser dans le protocole DHCP.

1.2.2 Hostapd

Hostapd est un package permettant la création d'un hotspot à l'aide Raspberry Pi.
L'installation se fait avec la commande suivante :

```
apt-get install hostapd
```

Chapitre 2

Communication et Sécurité

2.1 Mise en place des communications LoRa

Afin de faire communiquer nos deux raspberrys nous allons utiliser le LoRa et plus précisément la bibliothèque RadioHead qui nécessite elle même la bibliothèque bcm2835. Mais avant de les installer nous devons activer le bus PCI utilisé par le composant LoRa. Pour cela nous avons juste à éditer le fichier /RASPI/-boot/config.txt en modifiant les lignes de la façon suivante : “‘ Uncomment some or all of these to enable the optional hardware interfaces dtparam=i2c_arm=on dtparam=i2s=on dtparam=spi=on dtoverlay=gpio-no-irq “‘.

Nous installons ensuite nos bibliothèques :

```
$ wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.
  gz
$ tar zxvf bcm2835-1.71.tar.gz
$ cd bcm2835-1.71
$ ./configure
$ make
$ sudo make check
$ sudo make instal
$ cd ..
$ git clone https://github.com/hallard/RadioHead
```

Puis nous allons copier nos l'un des deux programmes rf95_server.cpp et rf_95_client.cpp dans le répertoire RadioHead/exemples/raspi/rf95. Le programme lié au serveur pour le Raspberry lié au serveur et inversement.

2.2 Mise en place du serveur mosquitto

— Sur le Raspberry lié au client LoRa précédemment configuré nous allons devoir installer mosquitto :

```
$ sudo apt-get install mosquitto
$ sudo apt-get install mosquitto-clients
```

```
```

— Puis nous éditerons le fichier /etc/mosquitto/mosquito.conf pour le transformer en :

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

#log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

allow_anonymous false
password_file /etc/mosquitto/mosquitto_passwd
```

— Nous allons par la suite créer un utilisateur ici "esp" avec pour mot de passe "tmctmctmc" :

```
  $ sudo mosquitto_passwd -c /etc/mosquitto/mosquitto_passwd
    esp
```

— Puis nous copions les fichiers tcp.conf et tls.conf fournit dans le git dans le répertoire /etc/mosquitto/conf.d/.

Le serveur mis en place il nous reste donc à configurer nos certificats pour cela nous allons juste à exécuter le script genCertif, préalablement placé dans le répertoire /client/home/pi/CA_ECC/

Voici notre script genCertif :

```
 openssl ecparam -out ecc.ca.key.pem -name prime256v1 -genkey
 openssl ecparam -out ecc.raspi.key.pem -name prime256v1 -genkey
 openssl ecparam -out ecc.esp.key.pem -name prime256v1 -genkey


openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\
   n[ext]\nbasicConstraints=CA:TRUE") -new -nodes -subj "/C=FR/L=
   Limoges/O=TMC/OU=IOT/CN=ACTMC" -x509 -days 3650 -extensions
   ext -sha256 -key ecc.ca.key.pem -text -out ecc.ca.cert.pem

openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\
   n[ext]\nbasicConstraints=CA:FALSE") -new -subj "/C=FR/L=
   Limoges/O=TMC/OU=IOT/CN=serveur.iot.com" -reqexts ext -sha256
   -key ecc.raspi.key.pem -text -out ecc.raspi.csr.pem

openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.
   key.pem -CAcreateserial -extfile <(printf "basicConstraints=
   critical,CA:FALSE\n\nsubjectAltName=DNS:localhost") -in ecc.
   raspi.csr.pem -text -out ecc.raspi2.pem
```

```
11
12 openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.
   key.pem -CAcreateserial -extfile <(printf "basicConstraints=
   critical,CA:FALSE") -in ecc.raspi.csr.pem -text -out ecc.raspi
   .pem
13
14
15 openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\
   n[ext]\nbasicConstraints=CA:FALSE") -new -subj "/C=FR/L=
   Limoges/O=TMC/OU=IOT/CN=capteur" -reqexts ext -sha256 -key ecc
   .esp.key.pem -text -out ecc.esp.csr.pem
16
17 openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.
   key.pem -CAcreateserial -extfile <(printf "basicConstraints=
   critical,CA:FALSE") -in ecc.esp.csr.pem -text -out ecc.esp.pem
18
19
20 sudo cp ecc.ca.cert.pem /etc/mosquitto/ca_certificates/ecc.ca.
   cert.pem
21 sudo cp ecc.raspi.key.pem /etc/mosquitto/ca_certificates/ecc.
   raspi.key.pem
22 sudo cp ecc.raspi.pem /etc/mosquitto/ca_certificates/ecc.raspi.
   pem
23 sudo cp ecc.ca.cert.pem /etc/mosquitto/ca_certificates/ecc.ca.
   cert.pem
24 sudo cp ecc.esp.pem /etc/mosquitto/certs/ecc.esp.pem
25 sudo cp ecc.esp.key.pem /etc/mosquitto/certs/ecc.esp.key.pem
26
27 sudo chmod 544 ecc.raspi.key.pem
```

Une fois cela fait nous démarrons le service mosquitto :

```
1   $ sudo systemctl enable mosquitto.service
2   $ sudo systemctl start mosquitto.service
```

En cas de problème de connexion il faudra exécuter le script setupRedirect depuis le Raspberry.

2.3 Mise en place de l'ESP

Pour commencer à mettre en place l'ESP nous allons premièrement installer mongoose-os et docker.io

```
1   $ sudo add-apt-repository ppa:mongoose-os/mos
2   $ sudo apt update
3   $ sudo apt install mos_latest
4   $ mos --help
5   $ sudo apt install docker.io
6   $ sudo groupadd docker
7   $ sudo usermod -aG docker $USER
```

Nous exécutons donc le script atecECC, nous sommes toujours dans le répertoire /TMCProject/. Notre esp est donc prêt a être utilisé. Nous devons vérifier et corriger si ce n'est pas le cas que notre certificat client est bien sous le format

Voici script atecECC :

```
openssl rand -hex 32 > slot4.key

sudo mos -X atca-set-key 4 slot4.key --dry-run=false

sudo mos -X atca-set-key 0 client/home/pi/CA_ECC/ecc.esp.key.pem
    --write-key=slot4.key --dry-run=false

sudo cp client/home/pi/CA_ECC/ecc.esp.pem my-app/fs/ecc.crt.pem
sudo cp client/home/pi/CA_ECC/ecc.ca.cert.pem my-app/fs/server_ca
    .pem
```

Chapitre 3

Déploiement de projet

Pour pouvoir lancer et démarrer notre projet nous devons commencer à installer 2 hôtes comme expliquer en partie 1. Sur la machine du serveur LoRa nous allons juste démarrer le script "script_boot_rpi" , puis nous connecter en ssh sur le Raspberry, installer notre serveur LoRa comme indiqué en partie 2. et exécuter la commande suivante dans le répertoire /RadioHead/exemples/raspi/rf95/

```
$ make rf95_server && sudo ./rf95_server
```

Notre serveur est désormais lancé et en écoute. Nous n'avons plus qu'à lancer le client LoRa pour cela sur la machine du client nous allons juste démarrer le script "script_boot_rpi" s'y connecter puis mettre en place le serveur Mosquitto et l'esp comme indiqué en partie 3 et 4. Une fois cela fait nous allons démarrer le script "script_ap" depuis le raspberry. Depuis l'hôte nous exécutons depuis le répertoire /my_app/ la commande suivante :

```
$ sudo mos build --local --platform esp8266 && sudo mos flash
  && sudo mos console
```

Notre esp devrait donc se connecter au réseau wifi créé par le Raspberry (scrit_ap). Il ne nous reste plus qu'à , depuis le Raspberry, exécuter la commande suivante dans le répertoire /RadioHead/exemples/raspi/rf95/

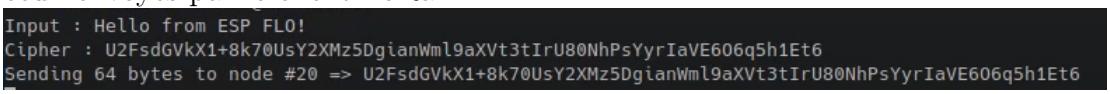
```
$ make rf95_client && sudo ./rf95_client
```

Notre projet est donc désormais fonctionnel.

Chapitre 4

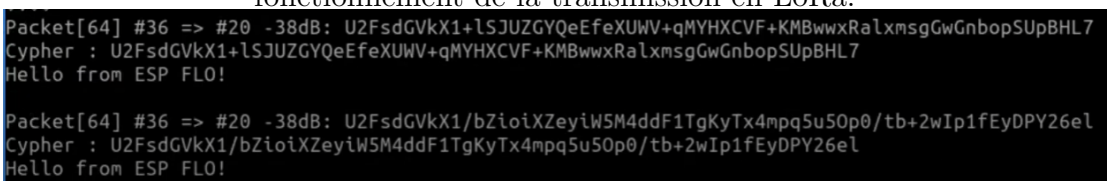
Démonstration

Afin d'observer le bon fonctionnement de notre mise en place du projet nous observons le client LoRa qui va lire les données envoyé par l'ESP sur le serveur Mosquitto, ce qui nous permet de garantir le fonctionnement de l'ESP et du serveur Mosquitto. Ici nous voyons bien les messages envoyés par l'ESP ainsi que ceux envoyés par le client LoRa.



Log du serveur LoRa

Puis nous pouvons consulter les logs du serveur LoRa afin de vérifier le bon fonctionnement de la transmission en LoRa.



Log du serveur LoRa

En effet nous pouvons y observer que le serveur reçoit un message chiffré qui une fois déchiffré nous dévoile le message original.