IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département informatique

Etude et réalisation d'une architecture MQTT

Rapport de stage de BUT informatique

LORIA

Mangin Adrien

Tuteur universitaire : Parmentier Yannick

Année universitaire 2024 -2025

IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département informatique

Etude et réalisation d'une architecture MQTT

Rapport de stage de BUT informatique

LORIA

Mangin Adrien

Tuteur universitaire : Parmentier Yannick

Année universitaire 2024 -2025

Remerciements

Je tiens à exprimer ma sincère gratitude à toutes les personnes qui ont contribué à la réussite de mon stage et à l'élaboration de ce rapport.

Je remercie tout d'abord mon maître de stage, M. Nataf, pour son accueil chaleureux, son encadrement bienveillant et les précieuses connaissances qu'il m'a transmises tout au long de ce stage.

Je souhaite également remercier l'ensemble de l'équipe SIMBIOT, et en particulier les doctorants, pour leur disponibilité, leur soutien et les échanges enrichissants qui ont grandement facilité mon intégration et l'accomplissement de mes missions.

Table des matières

Table des figures

Chapitre I : Introduction

Ce stage a été réalisé dans le cadre de ma deuxième année de Bachelor Universitaire et Technologique en Informatique à l'IUT (Institut Universitaire de Technologie) Nancy Charlemagne. Il s'est tenu du 17 février 2025 au 26 avril 2025, soit une durée totale de dix semaines. À travers ce stage, j'ai pu manipuler des protocoles de communication liés à l'Internet des Objets, les services Linux et leurs articulations avec des scripts bash. Ce stage fut également une opportunité de découvrir le monde de la recherche.

Ce rapport vise à présenter le sujet et son environnement dans un premier temps, puis s'intéressera aux différentes phases de réflexions et de développements qui se sont succédé afin de mener à bien le sujet du stage. Une dernière partie fera enfin état du projet à l'issue des dix semaines de stage afin de faire un bilan et une analyse du travail effectué, de ses modifications et extensions futures.

I.1 Présentation de l'entreprise

I.1.1 Loria

J'ai effectué mon stage au sein du ***LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications)**** situé à Vandœuvre-lès-Nancy et Villers-lès-Nancy. Il s'agit d'une ***UMR (Unité Mixte de Recherche)**** associant le ***CNRS (Centre national de la recherche scientifique)***, ***INRIA (Institut national de recherche en sciences et technologies du numérique)***, l'université de Nancy et ***CentraleSupélec****, ce qui en fait l'un des plus grands centres de recherche en informatique en France.

Le LORIA a été créé en 1997 et a pour mission la recherche fondamentale et appliquée en informatique. Il se compose de 28 équipes divisées en 5 départements dont 14 sont

communes avec INRIA*, ce qui représente plus de 500 personnes. Il est actuellement dirigé par M. TOUSSAINT Yannick. *(voir organigramme en Annexe Figure IV).*

I.1.2 Équipe SIMBIOT

J'ai été intégré dans l'équipe ***SIMBIOT**** puisqu'il s'agit de l'équipe de mon maître de stage. Cette équipe fait partie du département 3 : Réseaux, systèmes et services. Le nom de l'équipe a pour signification SIMulating and Building IOT, c'est-à-dire simuler et construire l'internet des objets. Elle se consacre à la conception et à l'évaluation de systèmes cyber-physiques intelligents (CPS). Ces systèmes se distinguent par un haut niveau d'autonomie, leur capacité d'adaptation, leur gestion de la complexité, ainsi que par des garanties en matière de sécurité, de sûreté et de confiance pour les utilisateurs humains. Les CPS étudiés par SIMBIOT reposent sur l'intégration de composants hétérogènes, aux interactions variées, évoluant dans des environnements dynamiques et partiellement non déterministes. Pour répondre à ces enjeux, l'équipe axe ses travaux sur l'intelligence adaptative des CPS, à travers deux fondations essentielles : le calcul distribué et les réseaux de communication. Les recherches sont structurées selon trois axes principaux : les abstractions et architectures pour la modélisation multi-niveaux et la conception modulaire ; les algorithmes pour le calcul distribué efficace ; et les architectures réseau, protocoles et mécanismes de qualité de service pour les interactions temps réel. Chaque défi est validé par des approches de co-simulation et d'expérimentation in situ, qui constituent un défi transversal à part entière.

I.2 Sujet

Ma mission se place dans le contexte d'un projet de recherche déjà existant : ***Simsennet****, un outil de mesure de la qualité de l'air constitué d'un réseau de capteurs de particules fines puis d'autres données par la suite. Ce réseau est couplé à un ordinateur ***Raspberry Pi*** qui récupère les données et les envoie sur un serveur afin qu'elles soient exploitées par la suite.

Ce projet est né de la collaboration entre l'équipe SIMBIOT et l'***INRS (Institut national de recherche et de sécurité)***. Ma mission est de revoir l'accès et la transmission des données, elle consiste à utiliser le protocole ***MQTT*** pour faciliter la trans-

mission des données puis d'intégrer un ***VPN*** pour permettre un accès à distance à des fins de maintenance et de récupération des données.

I.3 Problématique et objectifs

La problématique porte sur l'***étude et la réalisation d'une architecture MQTT**** cohérente qui répond aux besoins du projet. L'objectif de ma mission est de modifier l'architecture déjà présente pour rendre le projet plus accessible.

Un des objectifs principaux de Simsennet est de rendre l'outil accessible à des non-initiés. Un accès par VPN* mis en place contribue à cet objectif en permettant l'accès au système de manière simple depuis l'extérieur du réseau du LORIA.

Le chapitre suivant se concentre sur la mise en place de la nouvelle architecture et ses problématiques.

Chapitre II : Développement

II.1 Etude du projet : simsennet

II.1.1 Situation initiale

Mon stage se déroule dans le contexte d'un projet de recherche déjà existant : Simsennet. Il m'a donc fallu, dans un premier temps, comprendre le fonctionnement et l'architecture déjà en place.

Initialement, il s'agit d'un réseau de capteurs de microparticules qui permet de mesurer la qualité de l'air dans un souci de santé publique et de sécurité au travail. Ce réseau se compose de plusieurs Raspberry Pi* sur lesquels sont branchés les capteurs ; ils communiquent entre eux en déployant un réseau sans fil maillé privé. Les données collectées par les capteurs sont stockées sur un des ordinateurs qui fait office de passerelle. Cette passerelle est un point d'accès auquel doit se connecter un PC de contrôle local pour effectuer la récupération des données et les différentes opérations de gestion des machines du réseau.

Cette configuration, illustrée par la Figure II.1 est fonctionnelle, mais reste assez contraignante dans sa mise en place et sa maintenance. C'est donc dans l'objectif de faire évoluer cette architecture vers celle de la Figure II.2 que j'interviens. Je vais mettre en place les technologies mentionnées dans l'introduction afin de rendre Simsennet plus flexible.

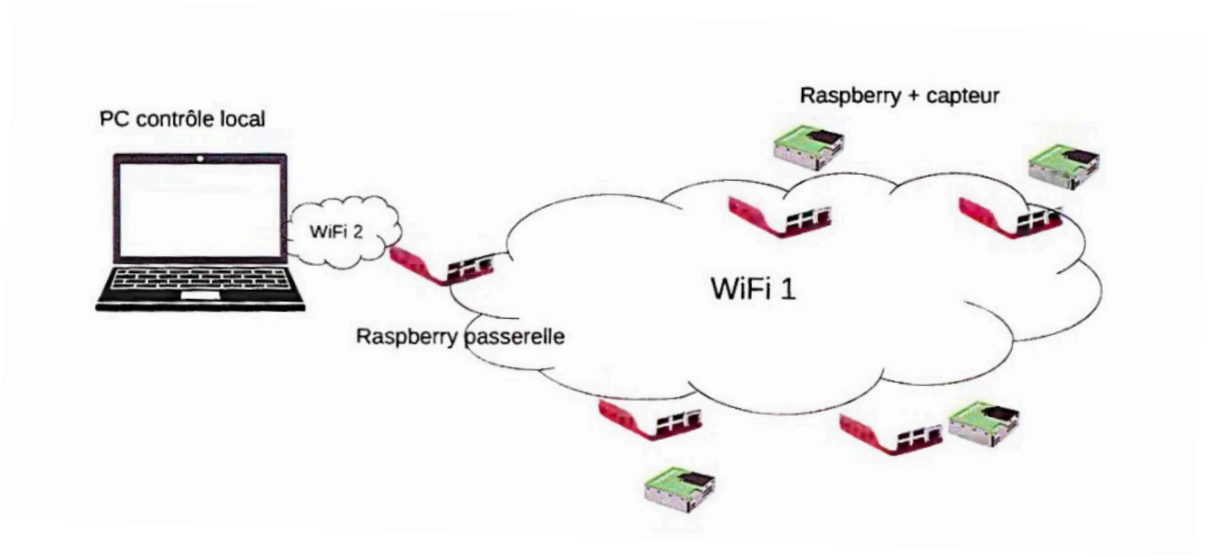


FIGURE II.1 – Illustration de la situation initiale

Cette architecture est en effet assez simple, la passerelle déploie un réseau wifi pour les nœuds et un second qui n'est accessible que par un pc en local. C'est relativement limitant car cela impose de devoir gérer Simsennet sur le lieu de l'expérience, en cas d'échec, impossible de relancer sans revenir sur ce lieu.

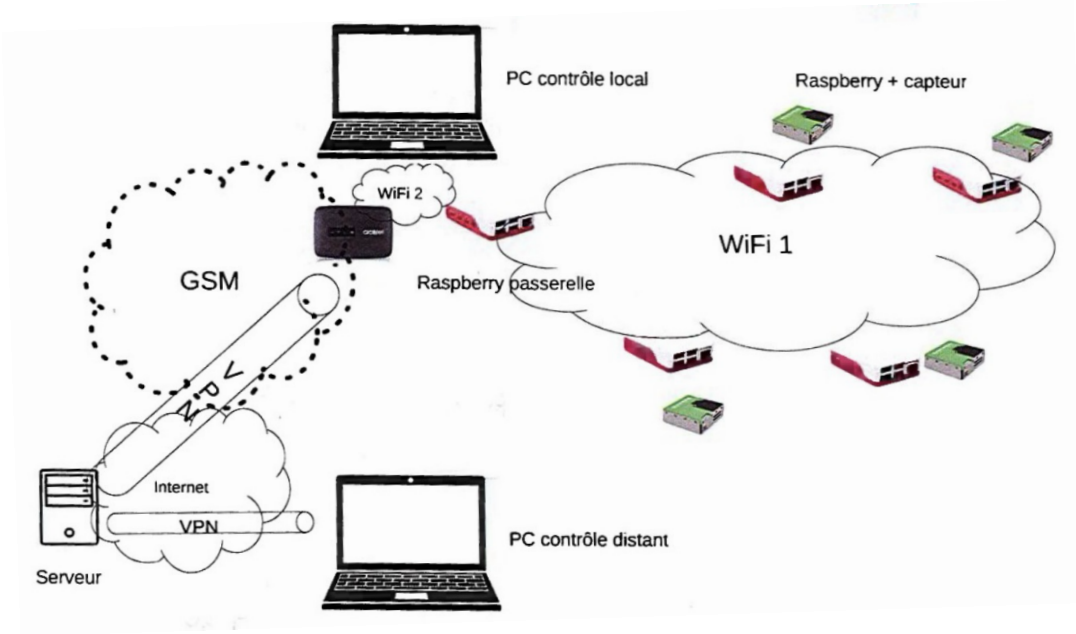


FIGURE II.2 – Illustration de la nouvelle architecture proposée

Cette deuxième architecture proposée est nettement plus pratique et résiliente. Le réseau des nœuds reste inchangé mais c'est sur la passerelle que le fonctionnement évolue. Elle dispose désormais d'un réseau wifi connecté à une antenne GSM. Cette antenne

dispose d'une carte SIM et d'un accès à Internet en 4G. La passerelle peut donc contacter tout appareil extérieur par ce biais. Comme sur le schéma, il est préférable d'utiliser un VPN pour contacter un serveur de manière sécurisé. Ce serveur est lui aussi accessible depuis une machine extérieure, ici le PC de contrôle distant, à partir d'un VPN.

II.1.2 Comprendre MQTT

Un des objectifs principaux est d'utiliser le protocole de communication MQTT*, j'ai donc dû me renseigner sur le fonctionnement de ce protocole.

Le protocole MQTT est un protocole fortement utilisé dans l'internet des objets, il fonctionne selon un principe de ***client-serveur**** et le principe d'abonné/éditeur (Publisher/Subscriber) sur différents canaux, appelés ***topics****.

Dans le contexte de Simsennet, un Raspberry Pi est par exemple (Figure II.3) abonné à un topic* global de mise à jour afin d'appliquer les changements de configuration, comme le changement de fréquence des mesures. Ainsi, le Raspberry Pi appliquera les modifications lorsqu'un changement de configuration est publié sur le topic de mise à jour par un éditeur comme le serveur. MQTT permet donc une communication efficace et ciblée entre les différents éléments qui constituent Simsennet.

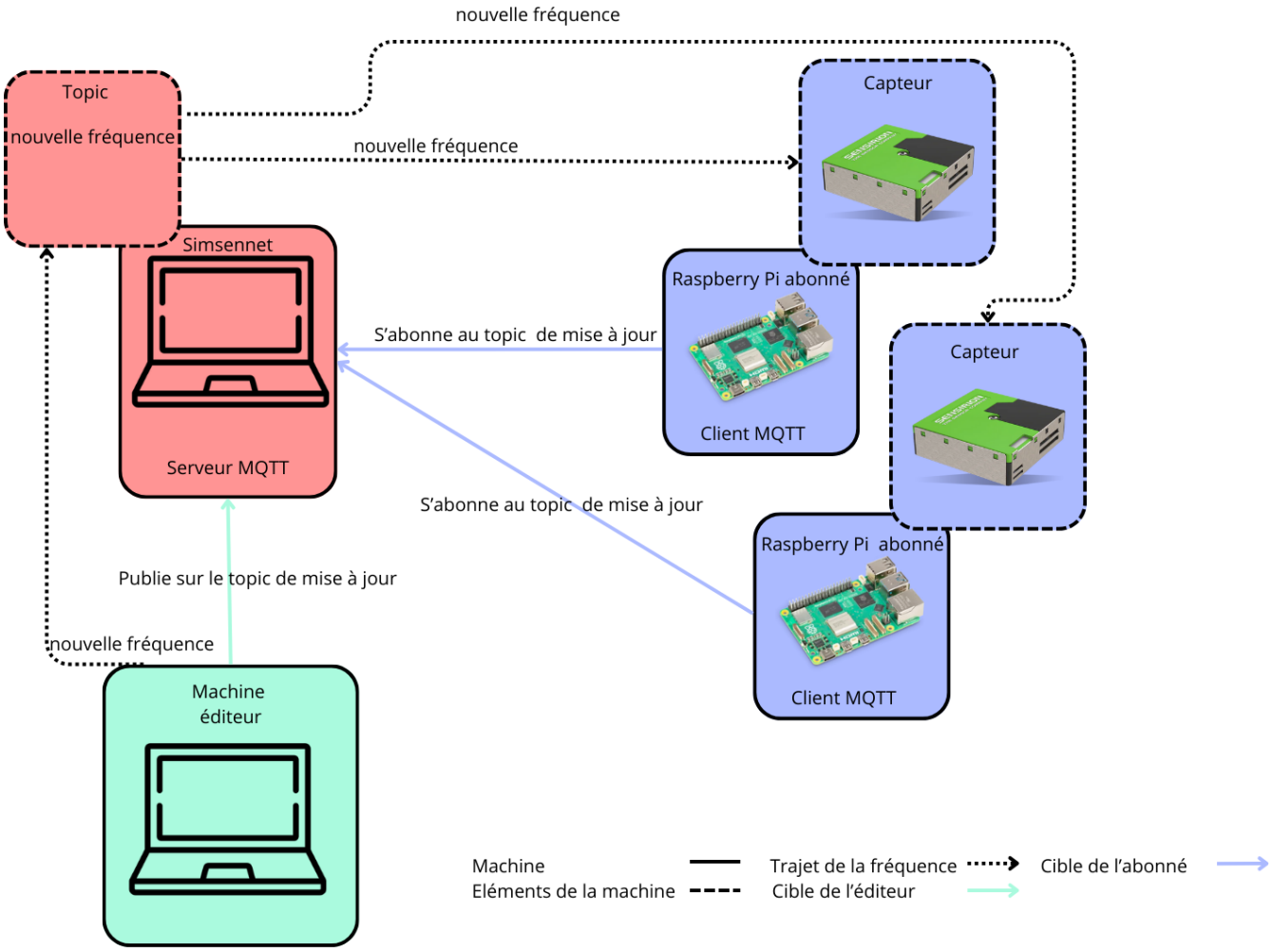


FIGURE II.3 – Illustration simplifiée de l'exemple

Dans cet exemple, on a un serveur MQTT en rouge sur lequel est défini un topic destiné à recevoir la fréquence. Lorsque la machine éditeur, symbolisée en vert, publie la fréquence souhaitée sur le topic de fréquence, elle apparaît alors sur celui-ci. Les deux Raspberry Pi encadrés en bleu, jouent le rôle d'abonnés, ils sont abonnés au topic de fréquence qui se trouve sur le serveur. Ainsi, lorsque une nouvelle fréquence est demandée, les deux capteurs la récupèrent depuis le topic et mettent à jour cette information.

II.1.3 Choix d'un broker MQTT

La mise en place du protocole MQTT se fait par le biais d'un broker qui coordonne les messages entre les différents clients. C'est l'outil qui utilise le protocole, il permet la communication entre les différents appareils qui l'utilisent. Le protocole étant répandu dans le domaine de l'internet des objets (***IoT****), il existe une multitude de brokers qui possèdent leur lot d'avantages et d'inconvénients. J'ai donc orienté mes recherches selon des critères simples : il me faut un broker : ***Open Source***, léger pour fonctionner sur les Raspberry Pi et facile d'utilisation.

Plusieurs solutions de brokers MQTT existent, chacune avec ses avantages et inconvénients. Mosquitto*, un broker open-source léger et performant, est populaire pour les petits projets et les systèmes embarqués, bien qu'il manque certaines fonctionnalités avancées de gestion des connexions à grande échelle. EMQX, également open-source, se distingue par sa capacité à gérer un très grand nombre de connexions simultanées et ses nombreuses fonctionnalités avancées (authentification, intégration cloud), mais il peut être plus gourmand en ressources. HiveMQ est une solution commerciale robuste, idéale pour les environnements d'entreprise nécessitant haute disponibilité et sécurité renforcée, mais son coût peut être un frein. Enfin, AWS IoT Core, un service géré par Amazon, offre une intégration native avec les autres services AWS, simplifiant l'évolutivité et la gestion, bien que sa dépendance à l'écosystème Amazon puisse limiter la flexibilité et engendrer des coûts récurrents. Le choix du broker dépend donc des besoins spécifiques en termes de performance, d'évolutivité et de coût.

Parmi ces brokers disponibles et répondant aux critères définis ci-dessus, ***Mosquitto*** m'a paru être le plus simple à prendre en main. Cet outil dispose en effet d'une documentation fournie et s'utilise en ligne de commandes, il est léger, ce qui est parfait pour une utilisation avec des Raspberry Pis. De plus, les fonctionnalités supplémentaires offertes

par EMQX ne sont pas nécessaires dans le cadre du projet Simsennet dont le nombre de nœuds ne pose pas de problème de communication. Mosquitto convient donc parfaitement au cas d'usage de ce projet.

II.1.4 Comprendre MOSQUITTO

Après avoir sélectionné un broker* il m'a fallu comprendre le fonctionnement de celui-ci afin de comprendre comment j'allais pouvoir l'intégrer dans le projet. Dans un premier temps, j'ai étudié son installation et sa configuration. Mosquitto peut fonctionner en tant que service sous différents systèmes d'exploitation (Linux, Windows) et propose une configuration simple via un fichier .conf. J'ai exploré les paramètres essentiels tels que l'activation de l'authentification, le chiffrement TLS et la gestion des accès. Dans un premier temps, le plus simple est d'autoriser les connexions anonymes pour éviter de gérer les accès. Je me permets cette approche qui est plus simple car l'accès au réseau est restreint et déjà sécurisé. De plus, le projet initial ne faisait pas non plus usage de telles mesures. En ce qui concerne le système d'exploitation, on utilise Raspbian, un dérivé DEBIAN qui est lui-même une version de Linux.

Ensuite, j'ai expérimenté la communication entre clients en utilisant les outils en ligne de commande (`mosquitto_pub et mosquitto_sub`). Cela m'a permis de tester la publication et la réception de messages, ainsi que de mieux comprendre la gestion des sessions et des Quality of Service (QoS). Cela m'a permis de tester la publication et la réception de messages en simulant différents scénarios de communication. Par exemple, j'ai expérimenté l'envoi de messages à des topics spécifiques en utilisant `mosquitto_pub`, puis leur réception en temps réel via `mosquitto_sub`. J'ai également observé le comportement des abonnements durables et temporaires, en redémarrant les clients pour analyser la persistance des connexions et des messages.

Cette exploration m'a également aidé à mieux comprendre la gestion des sessions dans Mosquitto. J'ai testé la différence entre les connexions avec et sans état (clean session = true/false) et constaté que, dans le cas d'une session persistante, un client pouvait récupérer les messages envoyés pendant son absence s'il utilisait le même client ID. Cela est particulièrement utile pour les dispositifs IoT qui peuvent se déconnecter temporairement mais doivent récupérer les messages manqués à leur reconnexion.

En parallèle, j'ai approfondi les Quality of Service (QoS), qui définissent le niveau de garantie de livraison des messages. En testant les trois niveaux disponibles :

- **QoS 0** (At most once) : le message est envoyé sans garantie de réception, ce qui est rapide mais peut entraîner des pertes en cas de déconnexion.
- **QoS 1** (At least once) : le message est garanti d'être livré au moins une fois, mais

peut être dupliqué si l'accusé de réception n'est pas bien géré.

— **QoS 2** (Exactly once) : le message est transmis exactement une fois grâce à un mécanisme de double validation, garantissant une livraison sans duplication, mais au prix d'une latence accrue.

En réalisant ces tests, j'ai pu identifier les niveaux de QoS les plus adaptés à mon projet en fonction des besoins en fiabilité et en performances. Par défaut, Mosquitto utilise une QoS de 0 ce qui convient à Simsennet puisque la fréquence d'envoi des données est peu élevée, ce qui limite le risque de perte. On préfère garder une latence faible pour assurer la réception et éviter de recevoir des doublons.

Enfin, j'ai exploré les mécanismes d'intégration avec mon projet, en testant l'utilisation de Mosquitto avec des langages comme Python et bash. Ces expérimentations m'ont permis d'identifier les bonnes pratiques et d'optimiser l'architecture de communication pour garantir un fonctionnement fiable et sécurisé du système. J'ai donc décidé de principalement utiliser bash afin de rester le plus proche possible du système. bash est directement disponible sur la plupart des systèmes Linux sans nécessiter d'installation supplémentaire. Contrairement à Python, qui requiert parfois des bibliothèques spécifiques (paho-mqtt par exemple), bash permet d'utiliser directement les commandes `mosquitto_pub et mosquitto_sub` sans dépendances supplémentaires. Cette approche garantit une exécution rapide et un faible impact sur les ressources système, ce qui est crucial pour les environnements embarqués ou les serveurs légers.

II.2 Tests dans le contexte de Simsennet

II.2.1 Evolution vers MQTT

Initialement, la communication au sein du réseau se fait avec un autre moyen de communication (netcat), et il faut définir une série de ports qui vont servir de point d'arrivée à un certain type d'information. Le port 16664 est, par exemple, configuré de manière à ne recevoir que la fréquence d'enregistrement. L'utilisation de Mosquitto et MQTT permet aisément de simplifier la configuration nécessaire à l'aide des topics qui sont plus versatiles et simples à configurer que les différents ports utilisés. Cependant, il m'a tout de même fallu réfléchir à l'architecture la plus adaptée au projet.

En effet, plusieurs choix valides se sont présentés :

— chaque nœud est à la fois un client et un serveur

— la passerelle fait office de serveur principal

La première approche est correcte mais elle peut être assez lourde puisqu'elle demande de gérer les différentes adresses des serveurs au niveau du fonctionnement global. Elle offre l'avantage d'être plus résiliente que la seconde qui présente un point de défaillance unique si le serveur de la passerelle venait à ne plus fonctionner. Subséquemment, la communication avec l'ensemble du système serait rendue impossible. Cependant, cette solution centralisée est plus simple à gérer.

II.2.2 Architecture centralisée et organisation des topics

J'ai donc opté pour la seconde approche qui, malgré le risque de défaillance, reste plus simple à gérer dans la majorité des cas. Je préfère en effet ne pas devoir gérer dynamiquement les différentes adresse IPs* et noms de machines qui peuvent varier dans mes scripts.

Une fois le choix de l'architecture client-serveur effectué, j'ai pu commencer à réfléchir à celle des topics. Le choix de l'architecture la plus simple permet également une gestion plus aisée des topics. Puisque chaque topic est relié à un seul serveur, il est plus facile pour moi de concevoir une hiérarchie cohérente et précise en évitant les confusions. J'ai décidé d'organiser les topics comme suit :

— **all** : définit la racine des autres topics.

— **pi** : définit le deuxième topic après la racine ; il symbolise le nœud.

— **data** : topic par lequel transitent les données, peu importe leur source.

— **sps30** : topic destiné aux données des capteurs de micro-particules (SPS30).

— **command** : topic destiné aux commandes de mise à jour de Simsennet.

On peut le voir comme une sorte d'arbre, dont la racine est le topic ***all***. La Figure II.4 permet de visualiser cette arborescence. Chaque Raspberry Pi peut aussi avoir des topics enfants qui lui sont propres.

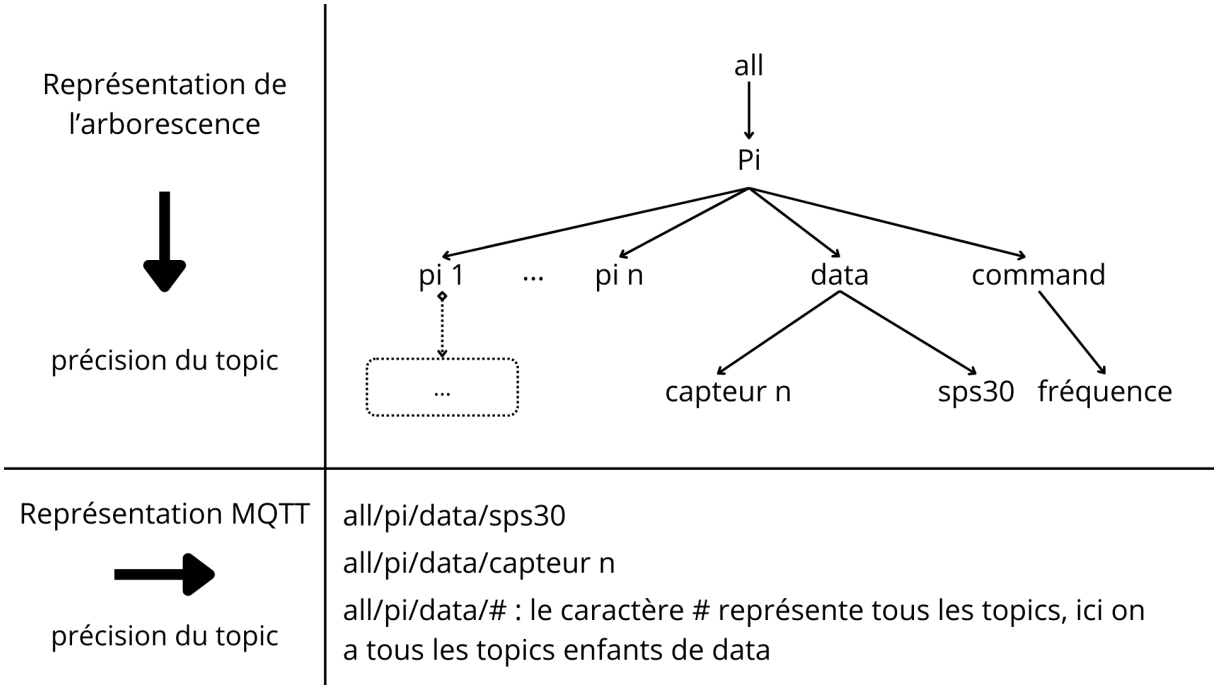


FIGURE II.4 – Illustration de l'arborescence des topics et leur représentation en MQTT

II.2.3 Test de l'architecture : problématique de charge réseau

Maintenant, que l'organisation et l'architecture sont choisies, il me faut confirmer que mes choix sont adéquats. Une de mes principales préoccupations est la notion de charge, je dois m'assurer que la communication ne soit pas impactée lorsque le nombre de Raspberry Pi est élevé. Un nombre conséquent de nœuds implique, en effet, que ceux-ci doivent pouvoir communiquer, parfois en même temps, et recevoir les informations en limitant au maximum les pertes.

En ce qui concerne la communication concurrente, le choix de Mosquitto comme broker MQTT s'avère une nouvelle fois pertinent. Léger et performant, Mosquitto est conçu pour gérer un grand nombre de connexions simultanées tout en maintenant une faible empreinte

mémoire et CPU. Il permet ainsi d'assurer une communication fluide et réactive, même lorsque de nombreux Raspberry Pi publient ou reçoivent des messages en parallèle. Sa capacité à maintenir la stabilité du flux de données sous forte charge contribue à garantir la fiabilité de l'architecture choisie.

De plus, il existe initialement un algorithme qui permet de définir la fréquence à laquelle les différents nœuds publient leurs données, appelé TDM* (Time Division Multiplexing). Je réutilise cet algorithme de manière à garder cette sécurité supplémentaire qui diminue davantage les risques de pertes de données et de surcharge. Cet algorithme se divise en deux parties, une partie client sur chaque nœud et une partie sur le serveur.

Le code suivant se trouve sur la passerelle et s'occupe d'envoyer la liste des nœuds du réseau à tous les nœuds. J'ai simplement modifié ce script afin d'utiliser les commandes mosquitto décrites précédemment. Dans le code, GW et TOPIC sont des variables, représentant respectivement la passerelle et le topic défini selon les choix d'architectures susmentionnés.

```
1  #!/bin/bash
2  GW="10.1.0.254"
3  TOPIC="all/pi/nodelist" # targets all sps30
4  if [ ! -d /var/run/simsennet ]
5  then
6     mkdir /var/run/simsennet
7  fi
8
9  cat /var/run/simsennet.leases | cut -d ' ' -f 4
10 | sort > var/run/simsennet/tdm.tmp
11
12 if [ ! -f /var/run/simsennet/tdm.run ] ||
13 ! cmp /var/run/simsennet/tdm.tmp /var/run/simsennet/tdm.run
14 &> /dev/null ; then
15    cp /var/run/simsennet/tdm.tmp /var/run/simsennet/tdm.run
16    nodelist=/var/run/simsennet/tdm.run
17    mosquitto_pub -h $GW -t $TOPIC -m "BEGIN"
18    mosquitto_pub -h $GW -t $TOPIC -f "$nodelist"
19    mosquitto_pub -h $GW -t $TOPIC -m "EOF"
20 fi
21 rm /var/run/simsennet/tdm.tmp
```

Réciproquement, on trouve un code similaire sur le client qui reçoit la liste des nœuds.

```
#!/bin/bash
TOPIC="all/pi/nodelist"
GW="10.1.0.254"
RUN_FILE="/var/run/simsennet/tdm.run"
TMP_FILE="/var/run/simsennet/tdm.tmp"
# Vérifier et créer le dossier si nécessaire
mkdir -p /var/run/simsennet
# Nettoyer le fichier temporaire
> "$TMP_FILE"
# Ecouter les messages MQTT
mosquitto_sub -h "$GW" -t "$TOPIC" | while read -r line; do
    # Détecter le début et la fin de la transmission
    if [[ "$line" == "BEGIN" ]]; then
    > "$TMP_FILE" # Réinitialiser le fichier temporaire
    elif [[ "$line" == "EOF" ]]; then
        # Si le contenu a changé, mettre à jour tdm.run
        if [ ! -f "$RUN_FILE" ]
        || ! cmp -s "$TMP_FILE" "$RUN_FILE";
        then
            mv "$TMP_FILE" "$RUN_FILE"
            echo "Fichier mis a jour : $RUN_FILE"
        fi
        # Vérifier si un timer est actif, sinon toucher le fichier
        if ! systemctl is-active
        "simsennet-[a-zA-Z0-9]*-sensor@[0-9]*.timer"
        &> /dev/null; then
            touch "$RUN_FILE"
            echo "Timer non actif, tdm.run mis à jour."
        fi
    else
        # Ajouter chaque ligne au fichier temporaire
        echo "$line" >> "$TMP_FILE"
    fi
done
```

Un deuxième script gère la liste reçue et définit l'ordre d'envoi à partir de cette liste. La méthode est assez simple, on parcourt la liste des différents nœuds reçus. Lorsque le nœud reconnaît son nom, il obtient son rang qui est égal à sa position dans la liste. L'heure de

départ est ensuite définie en divisant un intervalle de temps d'une minute par le nombre de nœuds qu'on multiplie par son rang. Cette méthode permet de répartir efficacement l'envoi sur une minute.

```
#!/bin/bash

if [ ! -f /var/run/simsennet/frq.run ];
then
  exit 0
fi
# ignore BEGIN
nodelist=$(cat /var/run/simsennet/tdm.run)
nodenb=0
noderank=-1
for h in $nodelist; do
  if [ $h == $(hostname) ]; then
      noderank=$nodenb
  fi
  nodenb=$(($nodenb + 1))
done
if [ $noderank -eq -1 ]
  exit 0
fi
if [ $nodenb -eq 0 ]
then # no node
  exit 0
fi
systemctl stop simsennet-sps30-sensor@[0-9]*.timer
calendar=$(( ($SSN_PERIOD$ / $nodenb)*$noderank ))
systemctl restart simsennet-sps30-sensor@$calendar.timer
```

Je n'ai pas rencontré de problème lié à la charge du réseau lors de mes tests avec 5 capteurs et une fréquence de mesure de 1 seconde. Bien que ce nombre ne soit pas nécessairement élevé, je pense pouvoir assurer que le système est suffisamment protégé contre un tel problème dans une utilisation réaliste de celui-ci.

II.3 Intégration à la solution existante

II.3.1 Vers une automatisation

Les tests confirment la capacité de l'architecture et des technologies employées, mais ce système requiert, jusqu'à présent, une mise en place manuelle. Or l'idée de ce projet est de fournir un outil simple d'utilisation et rapide à mettre en place. On souhaiterait un fonctionnement qui permet un déploiement comme celui illustré en Figure II.5. L'idée illustrée est effectivement relativement simple, l'utilisateur place ses Raspberry dans l'environnement de son choix afin d'y mener ses mesures. Il lance la passerelle puis chaque nœud de Simsennet après une trentaine de secondes, quelques minutes tout au plus, le système est opérationnel. C'est-à-dire que chaque nœud se tient prêt à recevoir les informations de fréquence, la liste de nœuds ou encore les données des capteurs dans le cas de la passerelle.

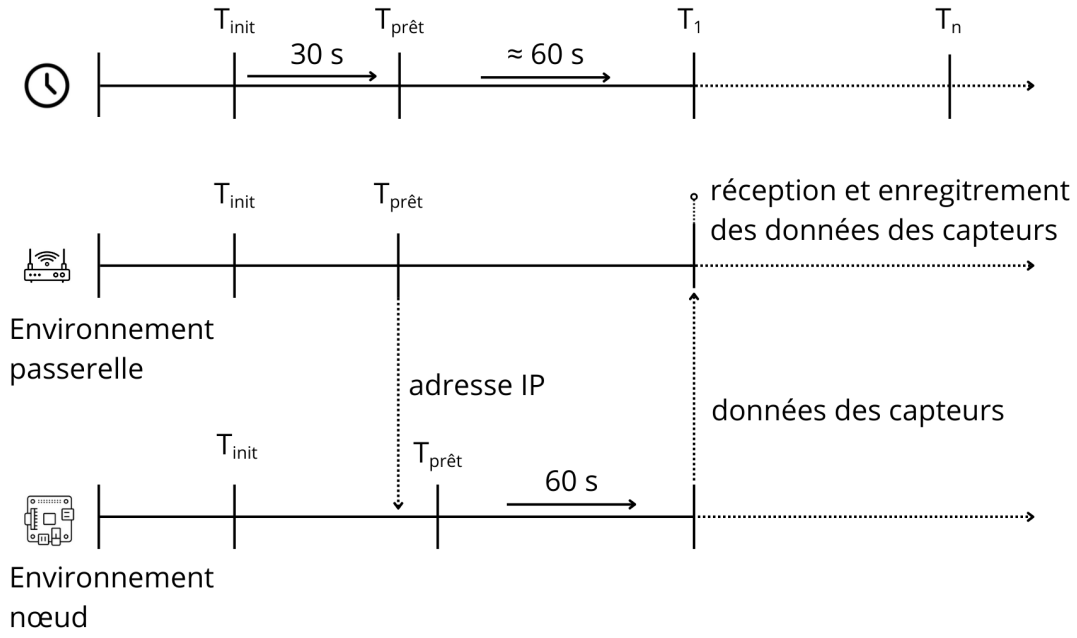


FIGURE II.5 – Illustration des étapes lors d'un déploiement

Le déploiement se fait en deux grandes phases, un premier temps d'initialisation assure que tous les éléments du réseau sont lancés. Ensuite, une durée s'écoule le temps que la passerelle constitue le réseau en attribuant une adresse IP à tous les nœuds, elle utilise pour cela un serivce DHCP*. Une fois qu'un nœud a reçu une adresse IP, il devient

opérationnel et se mettra à envoyer ses données toutes les minutes.

Un tel fonctionnement était déjà intégré dans le fonctionnement d'origine de Simsennet. En effet, un outil d'origine de Linux était déjà responsable d'automatiser l'ensemble des scripts. Cet outil, c'est Systemd.

II.3.2 Systemd

L'outil utilisé dans le processus d'automatisation étant parfaitement intégré au système d'exploitation Linux, je n'ai donc aucune raison de le changer. Je dois donc comprendre ce qu'est Systemd.

Systemd est un système d'initialisation (ou init system) et un gestionnaire de services pour les systèmes d'exploitation basés sur Linux. Il a été conçu pour remplacer les anciens systèmes d'init traditionnels, notamment SysVinit, afin d'améliorer la performance au démarrage, la gestion des processus, et la cohérence globale du système.

Systemd est aujourd'hui utilisé par la majorité des distributions Linux modernes comme Ubuntu, Debian et Raspbian.

Il repose sur plusieurs concepts clés :

— 1. Unités (Units)

Tout ce que systemd gère est représenté par une unité. Il en existe plusieurs types :

— **service** : pour les services (comme nginx.service)

— **socket** : pour le démarrage de service à la demande via des sockets

— **target** : pour grouper plusieurs unités (comme multi-user.target)

— **path** : pour surveiller les changement dans un fichier

— **timer** : pour des tâches planifiées (remplaçant cron)

device, path, etc.

Les fichiers unitaires sont placés dans /etc/systemd/system/ ou /lib/systemd/system/. Dans le cas du projet, c'est la seconde option qui prévaut.

— 2. Démarrage parallèle

Systemd lance les services en parallèle, ce qui réduit considérablement le temps de démarrage du système, parfait pour les Raspberry Pi.

— 3. Gestion fine des services

Systemd permet :

— Le redémarrage automatique d'un service en cas de crash (Restart=on-failure)

— Le contrôle des dépendances entre services

— La journalisation centralisée via journald à des fins de déboggage

— Le sandboxing des services pour améliorer la sécurité

En plus de créer les unités dont j'ai besoin, je manipule souvent systemd avec les commandes suivantes :

— systemctl status nginx.service → voir l'état d'un service

— systemctl start|stop|restart nginx.service → démarrer/arrêter/redémarrer un service

— systemctl enable nginx.service → activer un service au démarrage

— journalctl -xe → voir les logs système

— systemctl list-units –type=service → voir tous les services en cours

Systemd est donc une infrastructure complète pour la gestion du système Linux. Grâce à son intégration profonde au système d'exploitation, à sa gestion avancée des services et à sa capacité à automatiser et superviser de nombreuses tâches.

C'est donc bien l'outil idéal. L'idée est d'en faire une utilisation qui forme une sorte de chaîne d'unités qui se déclenchent les unes les autres. En général, une unité service lance un script, mais le service lui-même peut avoir été lancé par une autre unité portant le même nom que ce service. Une telle chaîne est décrite par la Figure II.6.

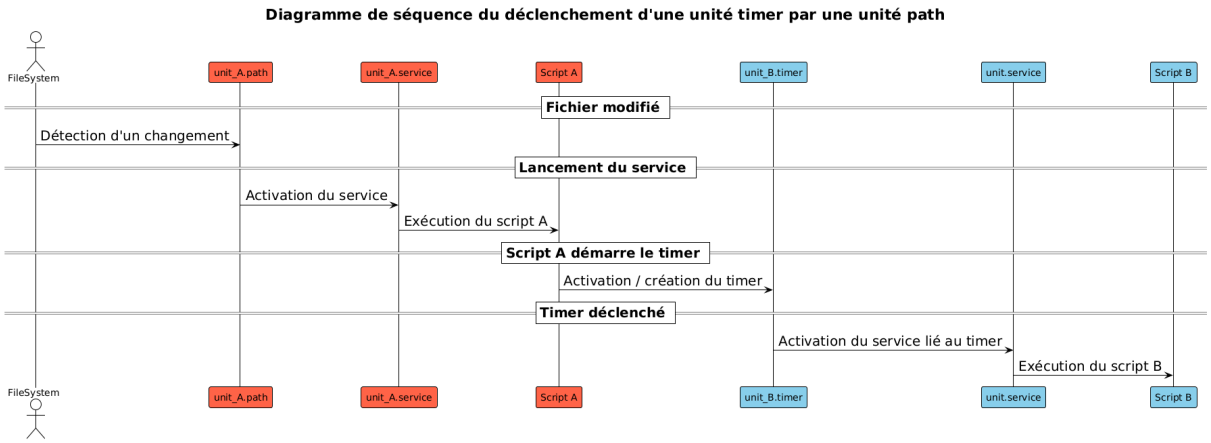


FIGURE II.6 – Illustration de la chaîne

Une unité path A surveille le contenu d'un fichier, lorsque ce fichier change, la chaîne se déclenche.

II.3.3 Articulations entre services

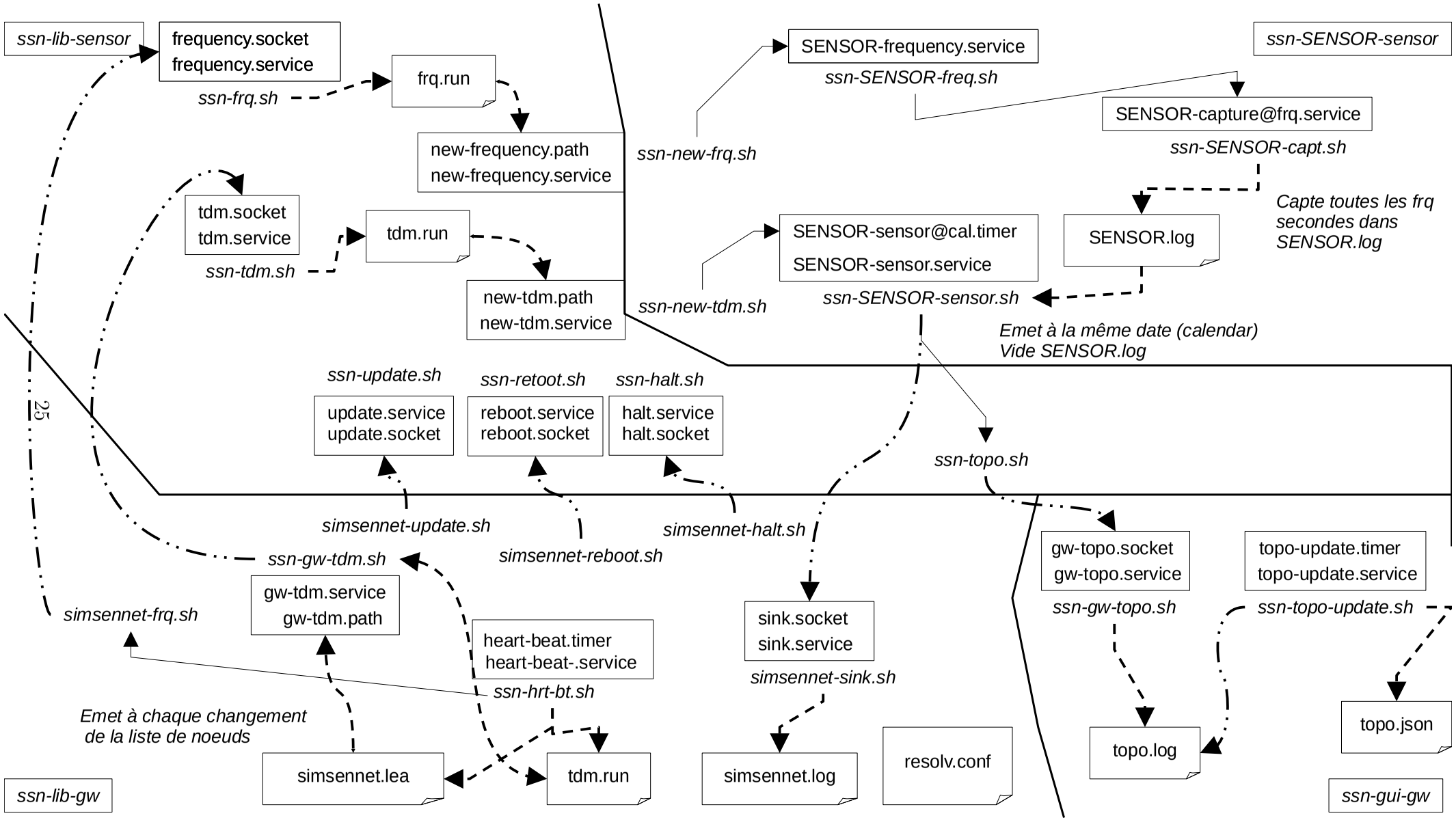
Maintenant que les architectures et la mise en place du système sont définies, je dois réadapter les articulations entre services et scripts pour correspondre à ma nouvelle architecture.

Le cœur de l'organisation initiale est basé sur les unités de type socket. Ces sockets sont configurés de manière à réagir aux informations arrivant sur les différents ports définis pour chaque type d'information. Un socket type se définit de la manière suivante :

```
1 [Unit]
2 Description = Simsennet Frequency socket
3
4 [Socket]
5 ListenDatagram=##SSN_FRQ_PORT##
6
7 [Install]
8 WantedBy = sockets.target
```

C'est dans la partie [Socket] que sont définis le port et le protocole utilisés, ici le protocole utilisé est l'UDP (User Datagram Protocol). Le port indiqué peut par exemple être celui attribué à la réception ou à l'envoi de la fréquence d'enregistrement des capteurs.

Pour pouvoir modifier l'état actuel de cette organisation d'unités Systemd, je dois d'abord en avoir une vue d'ensemble.


ssn-lib-sensor
frequency.socket
frequency.service
ssn-frq.sh
frq.run
new-frequency.path
new-frequency.service
ssn-new-frq.sh
SENSOR-frequency.service
ssn-SENSOR-freq.sh
ssn-SENSOR-sensor
SENSOR-capture@frq.service
ssn-SENSOR-capt.sh
Capte toutes les frq secondes dans SENSOR.log
SENSOR.log
tdm.socket
tdm.service
ssn-tdm.sh
tdm.run
new-tdm.path
new-tdm.service
ssn-new-tdm.sh
SENSOR-sensor@cal.timer
SENSOR-sensor.service
ssn-SENSOR-sensor.sh
Emet à la même date (calendar)
Vide SENSOR.log
ssn-update.sh
ssn-retoot.sh
ssn-halt.sh
update.service
update.socket
reboot.service
reboot.socket
halt.service
halt.socket
ssn-topo.sh
25
simsennet-update.sh
simsennet-reboot.sh
simsennet-halt.sh
ssn-gw-tdm.sh
gw-tdm.service
gw-tdm.path
simsennet-frq.sh
heart-beat.timer
heart-beat-.service
ssn-hrt-bt.sh
sink.socket
sink.service
simsennet-sink.sh
gw-topo.socket
gw-topo.service
ssn-gw-topo.sh
topo-update.timer
topo-update.service
ssn-topo-update.sh
topo.json
Emet à chaque changement
de la liste de noeuds
simsennet.lea
tdm.run
simsennet.log
resolv.conf
topo.log
ssn-lib-gw
ssn-gui-gw

Sur ce schéma, se distinguent trois zones, chacune délimitée par un contour. Le déclenchement de tout le processus se fait dans la partie inférieure gauche du schéma, encadré ssn-lib-gw. C'est en effet, de la passerelle que partent les informations essentielles que sont la fréquence et la liste de nœuds. Une unité timer est lancée de manière à envoyer périodiquement la fréquence aux nœuds, une unité path surveille le fichier des nœuds du réseau et renvoie ce même fichier lorsqu'un nœud s'y ajoute. Cela permet au réseau d'être à jour en cas de perte ou d'ajout d'un nœud. Ces informations de fréquence et de liste de nœuds sont transmises à tous les nœuds.

Le fonctionnement de ces nœuds est décrit dans la partie en haut à gauche, où se trouve l'encadré ssn-lib-sensor. C'est la partie relative aux unités et scripts génériques. Ce sont les éléments se trouvant sur chaque nœud. Les points de départ de cette chaîne sont les sockets et services du même nom, correspondant respectivement à la liste des nœuds et à la fréquence d'enregistrement. Une fois ces informations reçues, le relais est transmis à la partie supérieure droite du schéma.

Cette partie droite, symbolisée par l'encadré ssn-SENSOR-sensor, est responsable de la gestion d'un type de capteur particulier, sa construction est la même en ce qui concerne les unités, mais le contenu des scripts peut varier. Un capteur de CO2 ne fera pas appel aux mêmes instructions qu'un capteur de micro-particules. Les rôles des unités associés à un type de capteur sont multiples. Elles doivent récupérer la fréquence demandée afin de lancer la capture de données à la fréquence requise ainsi que de définir son rang dans la liste de nœuds pour connaître son horaire d'envoi.

Une fois cet horaire d'envoi connu, les données du capteur concerné reviennent vers la passerelle où le socket sink.socket s'occupe de les recueillir dans un fichier d'enregistrements log.

Enfin, la partie inférieure droite du schéma, encadré ssn-gui-gw concerne la partie graphique de Simsennet. Les unités de cette partie sont responsables de la réception de la topologie du réseau qu'elle reçoit des différents nœuds sur le socket dédié. Cela permet d'alimenter l'interface graphique de Simsennet qui est mise en place sur le serveur web lancé sur la passerelle.

Cette interface est accessible à l'adresse de la passerelle. Elle permet de télécharger les données des capteurs, de les mettre en pause ou encore de les arrêter. Il est aussi possible de changer la fréquence d'enregistrement de manière planifiée ou manuelle. La

zone centrale est réservée à l'affichage de la topologie avec une image de fond telle qu'un plan de bâtiment.

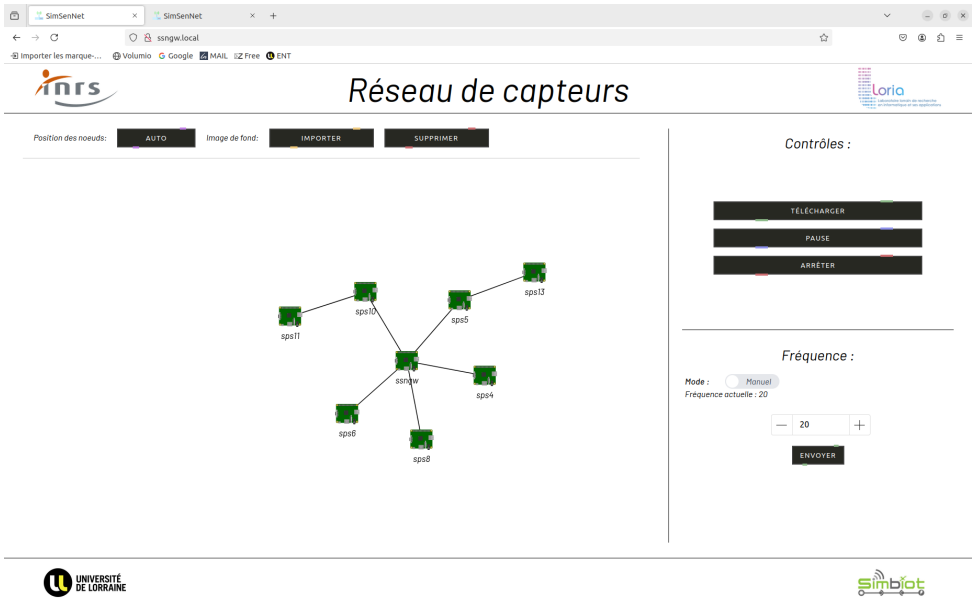


FIGURE II.7 – Capture d'écran d'une topologie obtenue sur l'interface graphique

On observe que certains Raspberry Pi se sont reliés de manière à faire partie du réseau malgré leur distance par rapport à la passerelle.

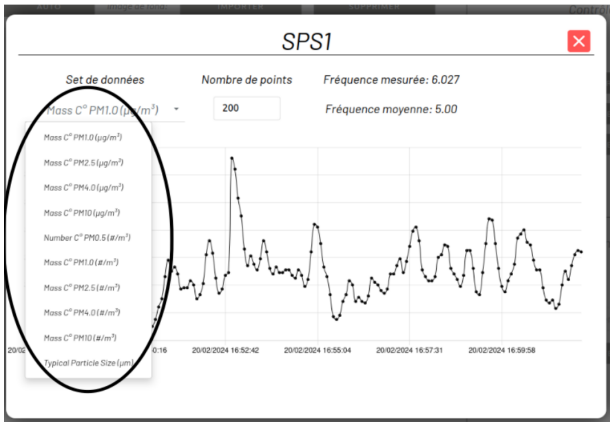
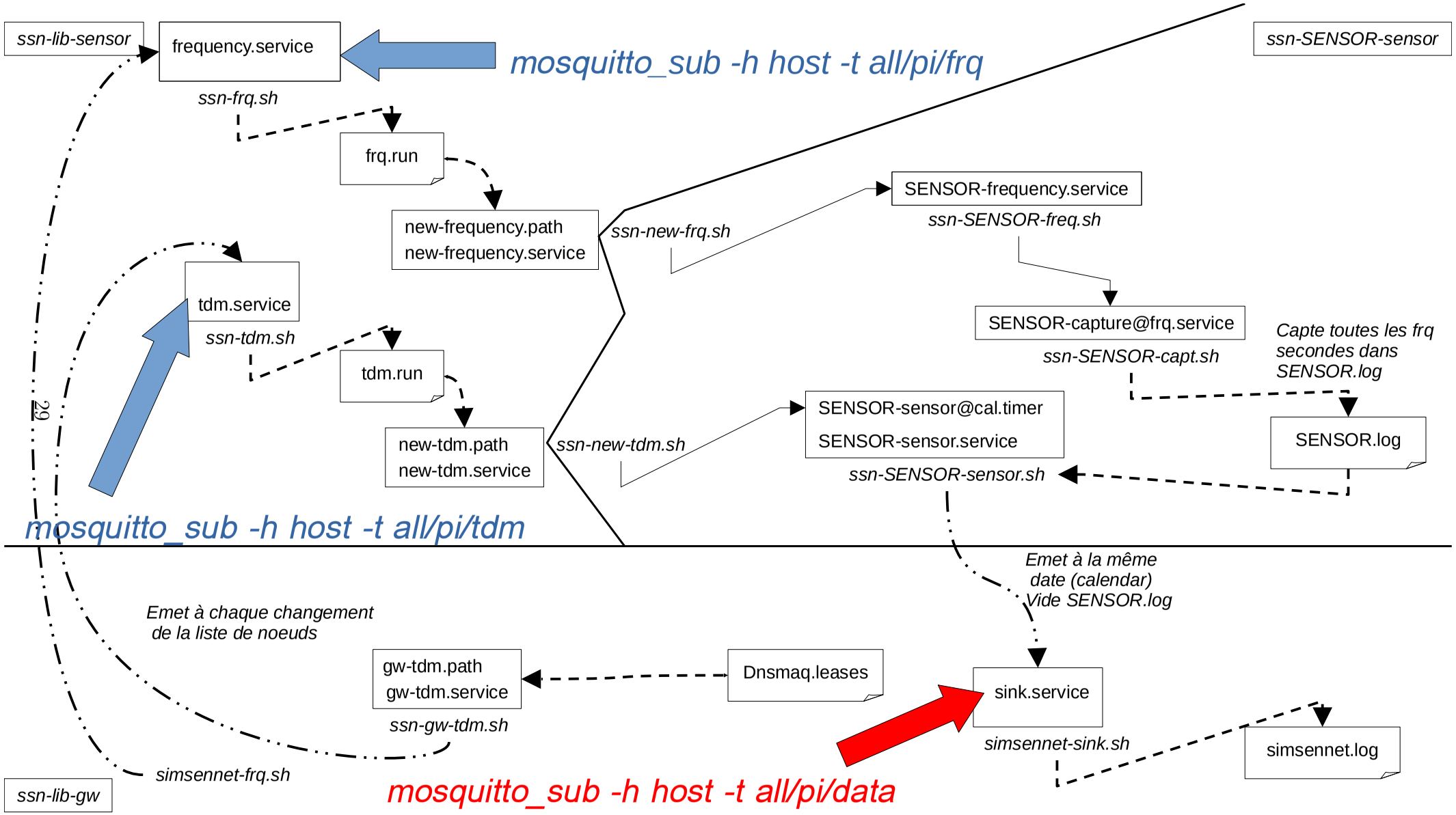


FIGURE II.8 – Capture d'écran de l'interface graphique. Zoom sur un nœud.

II.3.4 MQTT et articulations

Cette articulation entre les différents services est parfaitement fonctionnelle dans le cadre du fonctionnement initial de Simsennet mais utiliser MQTT m'oblige à retravailler ce fonctionnement.

Premièrement, comme mentionné plus tôt dans ce rapport, l'usage de MQTT me permet d'éviter l'usage de socket. Ce qui simplifie le schéma précédent.


ssn-lib-sensor
frequency.service
mosquitto_sub -h host -t all/pi/frq
ssn-SENSOR-sensor
ssn-frq.sh
frq.run
new-frequency.path
new-frequency.service
ssn-new-frq.sh
SENSOR-frequency.service
ssn-SENSOR-freq.sh
SENSOR-capture@frq.service
ssn-SENSOR-capt.sh
Capte toutes les frq secondes dans SENSOR.log
SENSOR.log
tdm.service
ssn-tdm.sh
tdm.run
new-tdm.path
new-tdm.service
ssn-new-tdm.sh
SENSOR-sensor@cal.timer
SENSOR-sensor.service
ssn-SENSOR-sensor.sh
mosquitto_sub -h host -t all/pi/tdm
Emet à la même date (calendar) Vide SENSOR.log
Emet à chaque changement de la liste de noeuds
gw-tdm.path
gw-tdm.service
ssn-gw-tdm.sh
Dnsmaq.leases
sink.service
simsennet-sink.sh
simsennet.log
simsennet-frq.sh
ssn-lib-gw
mosquitto_sub -h host -t all/pi/data

Globalement, tous les sockets ont été supprimés, les scripts qu'ils devaient lancer sont toujours là, mais fonctionnent différemment. Ces services lancent des scripts qui sont responsables d'écouter les messages MQTT, ils se basent sur le fonctionnement suivant :

```
mosquitto_sub -h $GW -t "$TOPIC" | while read frq; do
// instructions
done
```

Ils s'abonnent au topic qui remplace le port, tandis que la boucle while récupère les messages transmis sur le topic.

Mais d'autres problèmes se posent, l'utilisation de Mosquitto et MQTT apporte son lot de contraintes à ce système. En effet, Mosquitto fonctionne lui-même en tant que service. De plus, les scripts utilisant Mosquitto reposent sur le réseau maillé établi par la passerelle. Un nœud doit donc attendre que le service Mosquitto se soit lancé et qu'une adresse IP lui ait été attribuée. Sans quoi, les scripts utilisant Mosquitto rencontreront une erreur puisque, sans leur adresse, ils ne pourront pas contacter la passerelle.

Pour pallier cet éventuel problème, je peux définir pour chaque unité une notion de dépendance ou de départ selon une condition.

```
[Unit]
Description=Unité
After=mosquitto.service
Requires=mosquitto.service
```

Cela me permet d'éviter que ces services ne puissent se lancer sans Mosquitto. Par défaut, la plupart des unités sont activées automatiquement au démarrage, il faut donc désactiver les services afin d'éviter des états d'erreur. Lorsque l'on installe Simsennet on le fait à l'aide de paquets qui permettent l'installation de tous les fichiers nécessaires. Ces paquets permettent de définir un script qui s'exécute une fois à la suite de l'installation.

J'organise également mes scripts afin de ne déclencher les services que lorsqu'une adresse IP est attribuée. Sur la passerelle et les nœuds se trouve donc un script qui vérifie la présence d'une adresse IP puis lance le script suivant :

```
 #!/bin/bash

systemctl start mosquitto.service
systemctl start simsennet-networking.service
sleep 5
```

```
systemctl start dnsmasq.service
# parcourt des services
for service in $(systemctl list-unit-files | awk '{print $1}' | grep -E '^
    simsennet-.*\.service' | grep -vE '^simsennet-networking.service'| grep -vE '*
    @.service'); do
    echo "Restarting $service..."

    systemctl start "$service"

    if [[ $? -eq 0 ]]; then
        echo "$service restarted successfully."
    else
        echo "Failed to start $service."
    fi
done
# même logique pour les autres types d'unités
```

Ce script est responsable de la mise en route de la chaîne d'unités. Il lance Mosquitto puis le service réseau Simsennet avant de lancer les autres unités du projet. L'ensemble du système est désormais opérationnel sans interventions.

II.4 VPN et expérimentations

II.4.1 VPN : vers un déploiement en condition réelle

Maintenant que le système est opérationnel, il faut s'assurer qu'il soit utilisable dans le contexte d'une utilisation réelle, c'est-à-dire d'expérimentations sur le terrain. Pour cela, il est nécessaire qu'au-delà des contraintes matérielles que peut représenter l'utilisation de batteries ou celle d'une antenne wifi, l'infrastructure, notamment celle de l'interface graphique, soit accessible depuis l'extérieur. Je dois m'intéresser à la mise en place d'un VPN.

Un VPN (pour Virtual Private Network, ou Réseau Privé Virtuel en français) est une technologie qui permet de créer une connexion sécurisée entre un ordinateur (ou tout autre appareil) et un autre réseau via Internet. Grâce à un VPN, les données échangées sont chiffrées, ce qui empêche les tiers d'en lire le contenu. Dans le cadre de Simsennet, le serveur qui enregistre les données se trouve sur le réseau privé du laboratoire. Il faut

donc permettre à Simsennet de contacter le serveur d'enregistrement avec un VPN.

Cette fois-ci, nul besoin de me renseigner sur une quelconque solution de VPN, puisque la configuration et le programme sont déjà fournis par le service du Système d'Information.

Ce programme, c'est openconnect, un client VPN utilisable en ligne de commande que je n'ai qu'à intégrer dans un script sur la passerelle. L'utilisation est assez simple, une ligne suffit : `openconnect --user=UTILISATEUR --passwd-on-stdin -b vpn.inira.fr`. Je passe simplement mes identifiants ainsi qu'une option pour lancer le processus en tâche de fond à l'adresse du serveur vpn du laboratoire. Simsennet est désormais prêt à servir la science !

II.4.2 Expérimentations et déploiement sur le terrain

Il est maintenant temps de lancer une première expérimentation sur le terrain. Il serait intéressant de déployer l'ancienne version conjointement à celle-ci afin de s'assurer de la cohérence des mesures. Mon maître de stage, monsieur Nataf, a donc défini une date avant la fin de mon stage dans le but de mener une telle expérience.

Pour cette expérience, on met en place 7 Raspberry Pi pour chaque système. Chaque mini-ordinateur possède un capteur de micro-particules SPS30 et est alimenté par une batterie. Le dispositif est illustré sur la Figure II.9 et la Figure II.10.

Cette expérience se déroule à l'ENSIC à Nancy. On dispose les différents dispositifs au sein du bâtiment, en gardant un de chaque version que l'on place à l'extérieur à des fins d'étalonnage et de contrôle des mesures. La photographie prise lors de l'expérience montre deux Raspberry Pi nœuds de Simsennet. Ils sont placés sur un trépied et maintenus par un support imprimé en 3D.

On peut ensuite vérifier que l'ensemble des Raspberry Pi déployé a bien rejoint le réseau de Simsennet. Il ne resta qu'à laisser les capteurs tourner puis récupérer les différents résultats à l'aide de l'interface graphique.

FIGURE II.9 – Dispositif comprenant un Raspberry Pi, une batterie et un capteur de micro-particules SPS30

FIGURE II.10 – Simsennet déployé lors de l'expérience

Chapitre III : Conclusion

III.1 Simsennet

III.1.1 Aboutissement du projet

Après cette première expérience, je peux conclure qu'en ce qui concerne le sujet du stage lui-même, je l'ai mené à bien. En effet, la transition de Simsennet vers le protocole MQTT est effectuée et assure les mêmes fonctionnalités qu'auparavant. J'ai la satisfaction d'avoir achevé cette mission et d'avoir en quelque sorte contribué à la recherche.

III.1.2 Difficultés techniques rencontrées

J'ai bien sûr rencontré quelques difficultés techniques sur le projet. En effet, travailler sur des Raspberry Pi pose son lot de problèmes.

Je travaille sur ces ordinateurs sans interface graphique, ce qui peut faire peur au premier abord, mais c'est surtout une limitation contraignante dans la gestion de la machine. Je ne dispose que d'un unique terminal, impossible d'avoir des fenêtres supplémentaires pour déboguer en parallèle.

S'ajoute à cela le fait de travailler sur plusieurs machines indépendantes sur lesquelles il faut apporter les modifications manuellement. Je m'arrange pour devoir le faire le moins possible, mais fatalement, cela finit par arriver. Il arrive aussi que le code doive changer drastiquement lors des phases de tests, ce qui m'oblige à réinitialiser les machines de zéro et à réinstaller tout le système.

Parfois, je rencontre aussi des difficultés à identifier les sources de certains comportements inattendus. Je travaille effectivement avec une succession d'unités systemd décrites plus tôt, mais il arrive que je commette une erreur dans la logique de cette chaîne. Une erreur qui, bien souvent, fait sauter un maillon de cette chaîne sans que je m'en aperçoive,

ce qui a pour effet de saboter le système dans son ensemble. D'autres fois, c'est une boucle qui se crée et les unités se relancent incessamment entre elles. Pour déceler ces problèmes, je dois alors mener l'enquête dans les différents logs du système pour comprendre où se trouve l'erreur.

Enfin, il y a des erreurs qui proviennent des outils utilisés eux-mêmes, de certaines subtilités qui ne se révèlent que dans des situations particulières que sont celles des tests. Des conflits entre les configurations que j'ajoute et celles déjà présentes sur le système, des erreurs dans les droits d'accès des fichiers de configuration. Des erreurs matérielles, une carte réseau manquante, un capteur défectueux.

Toutes ces difficultés m'ont parfois fait perdre une quantité considérable de temps et mon sang-froid, mais elles m'ont permis d'appréhender mieux un système complexe tel que Simsennet où il faut développer sur des appareils différents. C'est un autre type de projet et de développement, conception que ce sur quoi j'avais l'habitude de travailler avant ce stage.

III.1.3 Évolution de Simsennet

En ce qui concerne l'avenir de Simsennet, mon travail avec l'implémentation d'un nouveau protocole de communication devrait permettre de faire de ce projet une base solide sur laquelle il devrait être facile de construire. En effet, le fonctionnement des topics permet d'en ajouter de nouveaux et ainsi d'apporter de nouvelles fonctionnalités aisément. L'intégration du VPN permet aussi de simplifier l'accès aux données récoltées.

III.2 Bilan

III.2.1 Compétences développées

Les outils et les difficultés techniques rencontrés m'ont permis d'acquérir de nouvelles compétences techniques. Je maîtrise désormais les unités systemd qui sont responsables du fonctionnement de beaucoup de systèmes Linux. J'ai renforcé mes compétences en création de scripts bash, qui sont un moyen simple d'automatiser des tâches sur un système. J'ai aussi acquis des compétences en réseau, en manipulant différents services comme le DNS* DHCP et leur configuration. Avec l'utilisation de MQTT et de Raspberry Pi, j'ai aussi

pu développer des compétences dans le domaine de la domotique et de l'internet des objets. La rédaction de ce rapport m'a aussi permis d'apprendre le LaTeX, un langage et un système de composition de documents qui sont utilisés dans le milieu de la recherche scientifique.

III.2.2 Découverte de la recherche et environnement

Ce stage au sein du LORIA fut aussi une opportunité pour moi de découvrir le fonctionnement de la recherche. J'ai pu approcher la recherche aux côtés de chercheurs et de doctorants, d'en apprendre plus sur les parcours qui mènent au doctorat, ainsi que sur les exigences du doctorat.

J'ai été rapidement et pleinement intégré à l'équipe SIMBIOT, réunion d'équipe et même sorties, je ressors de ce stage en ayant beaucoup appris sur le plan technique et humain.

Bibliographie

[1] ArchWiki contributors. ***systemd***. Consulté le 22 avril 2025. 2025. URL : https://wiki.archlinux.org/title/Systemd.

[2] Jason Bauer. ***How to Choose an MQTT Broker : Mosquitto vs HiveMQ vs Home Assistant***. Accessed : 2025-04-22. 2023. URL : https://efundies.com/how-to-choose-an-mqtt-broker-mosquitto-vs-hivemq-vs-home-assistant/.

[3] HiveMQ Team. ***HiveMQ vs. Mosquitto : An MQTT Broker Comparison***. Accessed : 2025-04-22. 2024. URL : https://www.hivemq.com/blog/hivemq-vs-mosquitto-an-mqtt-broker-comparison/.

[4] John. ***How to Create a Local Self-Hosted MQTT Mosquitto Broker on a Raspberry***. Accessed : 2025-04-22. 2022. URL : https://www.howtoraspberry.com/2022/04/how-to-run-mqtt-mosquitto-on-a-raspberry/.

[5] Roger A. Light. "Mosquitto : server and client implementation of the MQTT protocol". In : ***Journal of Open Source Software*** 2.13 (2017), p. 265. DOI : 10.21105/joss.00265. URL : https://joss.theoj.org/papers/10.21105/joss.00265.pdf.

[6] S. S. Manvi et P. V. Patil. "MQTT based home automation system using ESP8266". In : ***2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)***. 2017, p. 1736-1739. DOI : 10.1109/ICECDS.2017.8389760. URL : https://ieeexplore.ieee.org/abstract/document/7906845.

[7] ***MQTT Protocol Specification***. https://upfiles.heclouds.com/123/ueditor/2016/07/14/184e2dd5bc35bd9de59abc740665faac.pdf. Accessed : 2025-04-22. 2016.

Glossaire

Terme	Description	Pages
adresse IP	Numéro d'identification unique attribué à chaque appareil connecté à Internet.	16, 30
broker	Serveur MQTT, courtier, système back-end qui coordonne les messages entre les différents clients.	12, 14
CentraleSupélec	Grande école d'ingénieurs française.	5
client-serveur	Le protocole client–serveur désigne un mode de transmission d'information entre plusieurs programmes ou processus : l'un, qualifié de client, envoie des requêtes ; l'autre, qualifié de serveur, attend les requêtes des clients et y répond. Le serveur offre ici un service au client.	11
CNRS	Centre national de la recherche scientifique.	5
DHCP	Dynamic Host Configuration Protocol est un protocole réseau dont le rôle est d'assurer la configuration automatique des paramètres IP d'une station ou d'une machine, notamment en lui attribuant automatiquement une adresse IP et un masque de sous-réseau.	21, 36

Terme	Description	Pages
DNS	Domain Name System ou DNS est un service informatique distribué qui associe les noms de domaine Internet avec leurs adresses IP ou d'autres types d'enregistrements.	36
INRIA	Institut national de recherche en sciences et technologies du numérique.	5, 6
INRS	Institut national de recherche et de sécurité.	6
IoT	Internet of Things : Internet des objets, réseau d'objets et de terminaux connectés équipés de capteurs (et d'autres technologies) leur permettant de transmettre et de recevoir des données entre eux et avec d'autres systèmes.	12
IUT	Institut Universitaire de Technologie.	5
LORIA	Laboratoire Lorrain de Recherche en Informatique et ses Applications.	5, 7, 37
Mosquitto	Broker MQTT Open Source.	12, 13, 30, 31
MQTT	Protocole de messagerie publish-subscribe basé sur le protocole TCP/IP. Il est l'un des protocoles clés dans le déploiement de l'internet des objets.	6, 7, 11, 12, 28, 30, 35, 36
Open Source	Code conçu pour être accessible au public : n'importe qui peut voir, modifier et distribuer le code à sa convenance.	12
Raspberry Pi	Nano-ordinateur monocarte.	6, 8, 11, 12, 17, 22, 27, 32, 35, 36
SIMBIOT	Equipe de recherche du département 3 : Réseaux, systèmes et services, chargée de la conception et validation de système Cyber-Physiques.	6, 37

Terme	Description	Pages
Simsennet	Projet de recherche commun entre l'équipe SIMBIOT et l'INRS : réseau de capteurs de mesure de la qualité de l'air.	6–9, 11, 13, 15, 21, 22, 26, 28, 30–32, 35, 36
TDM	Time Division Multiplexing. Algorithme responsable de la répartition des envoies des données sur une minute.	18
topic	Dans le cadre de MQTT, structure arborescente hiérarchique, les subscribers MQTT s'abonnent aux messages associés à un topic donné, ces messages sont envoyé par les publishers..	11, 12, 16, 36
UMR	Unité Mixte de Recherche.	5
VPN	Virtual Private Network : Réseau Privé Virtuel qui assure l'anonymat, la confidentialité et la sécurité des informations échangées en ligne, par leur circulation chiffrée à l'intérieur d'un réseau public (Internet, notamment).	7, 10, 31, 32, 36

Chapitre IV : Annexes

ORGANIGRAMME

Date de mise à jour : 01/04/2025

DIRECTION / COMITÉ DE DIRECTION

Directeur : **Yannick Toussaint**
Directeurs adjoints : **Sylvain Lazard ; Armelle Brun**
Secrétaire générale : xxx
Assistante de direction : **Anne-Florence Remy**

CONSEIL DE LABORATOIRE

CONSEIL SCIENTIFIQUE

AREQ - Assemblée des REsponsables d'éQuipes

COMITÉS et COMMISSIONS

CLHSCT CUMI*
Mentorat
CARE : **S. Nowakowski**
Parité & Egalité : **M. Baron**
Ecoute de Proximité : **M. Duflot-Kremer, A. Dutech**

SERVICE DE GESTION LORIA

Resp. *xxx*

*** Pôle Budget**
Julie Buquet (référente de pôle)
Nathalie Fritz
Minouna Ziamni

*** Pôle Ressources Humaines**
Sylvie Hilbert (référente de pôle)
Gallown Nizard

*** Pôle Gestion**
Antoinette Courrier
Delphine Hubert
Nathan Grandemange
Elsa Maroko
Nathalie Béthus
Anne-Marie Messaoudi
Manon Consigny

SERVICE INFORMATIQUE de SOUTIEN à la RECHERCHE

Resp. Adrien Guenard

Jonathan Alcuta
Antoine Falcone
Patrice Ringot
Fabrice Sabatier
Sjoerd De Vries
Romain Karpinski
Cyril Regan
Mickaël Delcey
A venir (2 IR)

SERVICE INNOVATION TRANSFERT

Resp. Adrien Guénard

Julie Buquet
Sébastien Wald

SERVICE COMMUNICATION, MEDIATION et WEB

Resp. Marie Baron

Mercedes Rodriguez (projet web)
Mariana Diaz-Ramirez

Assistant de prévention
Antoine Falcone

SERVICE INFORMATIQUE/DSI

Référent SI. J-M. Antoine

Patrick Aubier, centre de service
Jean-Claude Giese service production*
Lionel Maurice
David Mazé, centre de service*

Personnel INRIA en italique
Personnel en CDD

(*) commun avec le centre INRIA de l'Université de Lorraine

28 EQUIPES – 5 DEPARTEMENTS

Équipes de recherche communes avec le Centre Inria de l'Université de Lorraine

Département 1. Algorithmique, calcul, image et géométrie

Responsable Pierrick Gaudry

- ABC – Y. Guermeur
- ADAGIO – I. Debled-Rennesson
- CARAMBA – E. Thomé
- TANGRAM – M-O. Berger
- MFX – S. Lefebvre
- GAMBLE – G. Moroz
- PIXEL – D. Sokolov

Département 2. Méthodes formelles

Responsable Jannik Dreier

- CARBONE – J-Y. Marion
- TYPES – D. Galmiche
- MOSEL – D. Méry
- MOCQUA – S. Perdrix
- PESTO – S. Kremer
- VERIDIS – S. Merz

Département 3. Réseaux, systèmes et services

Responsable Bernadetta Addis

- OPTIMIST – A. Oulamara
- SIMBIOT – V. Chevrier
- COAST – C. Ignat
- RESIST – I. Chrisment

Département 4. Traitement automatique des langues et des connaissances

Responsable Fabien Lauer

- K – M. D'Aquin
- ORPAILLEUR – M. Couceiro
- SMArt – K. Smaïli
- SYNALP – C. Cerisara
- MULTISPEECH – S. Ouni
- SEMAGRAMME – P. De Groote

Département 5. Systèmes complexes, intelligence artificielle et robotique

Responsable Alain Dutech

- BISCUIT – B. Girau
- BIRD – A. Brun
- NEURORYTHMS – L. Bougrain
- CAPSID –I. Chauvot de beauchêne
- LARSEN – F. Colas

(équipe) équipes et plateforme ZRR

AXES TRANSVERSES

Cybersanté : **E. Kerrien**
E-éducation : **I. Debled-Rennesson**
Sécurité : **M. Minier**
Systèmes cyberphysiques :
TAL et IA : **C. Cerisara**
Énergie : **V. Chevrier**
Usine du futur : **G. Simon**

LES PLATEFORMES

- CRÉATIV'LAB
- GRID 5000
- LHS
- MBI-DS4H

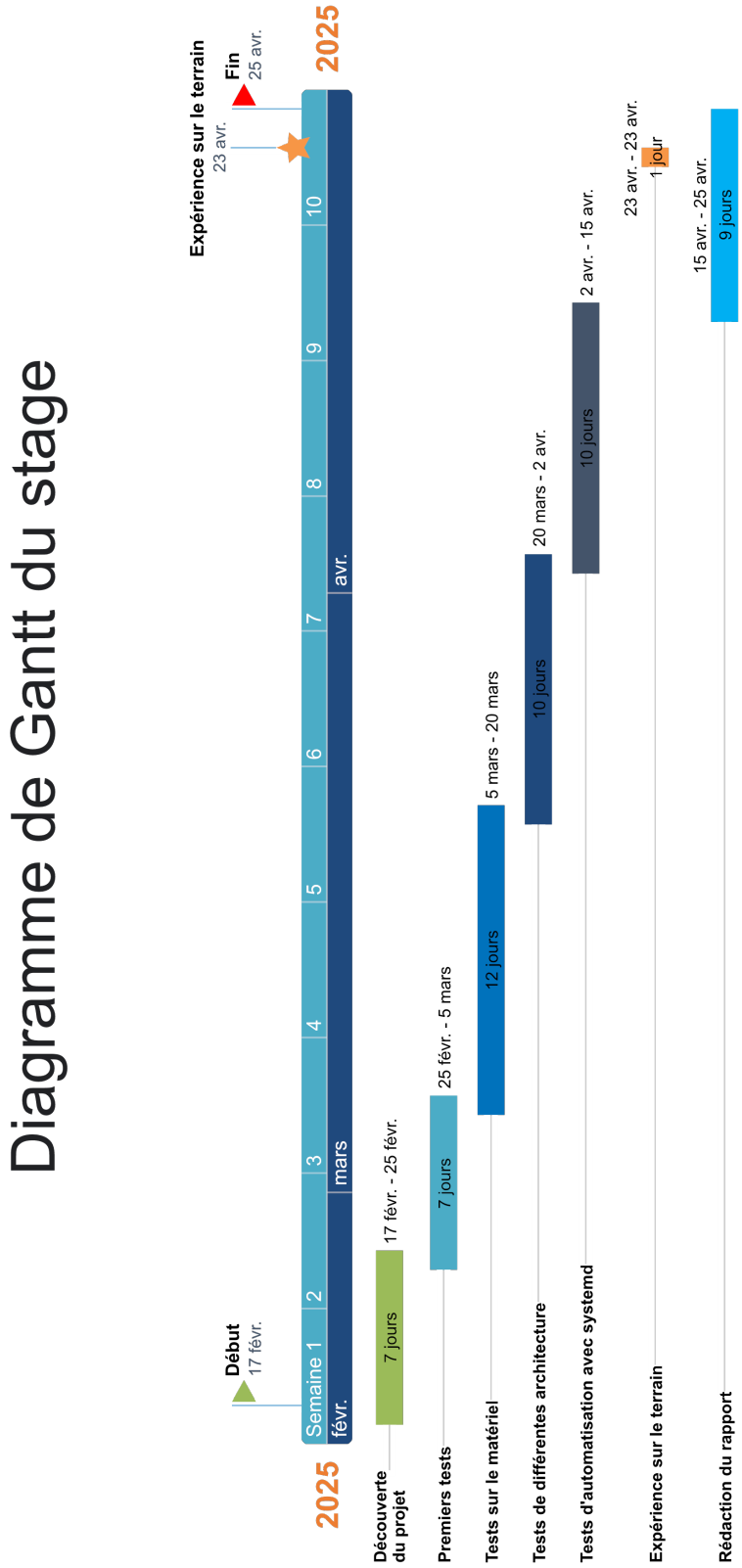


FIGURE IV.1 – Diagramme de Gantt du déroulement du stage

FICHE RAPPORT DESTINÉE À LA BIBLIOTHÈQUE

RAPPORT CONFIDENTIEL ET NE DEVANT PAS FIGURER À LA BIBLIOTHÈQUE :

Oui **Non**

NOM ET PRÉNOM DE L'ÉTUDIANT : MANGIN Adrien

BUT 2 INFORMATIQUE **S4**

BUT 3 INFORMATIQUE **S6**

TITRE DU RAPPORT : Étude et réalisation d'une architecture MQTT

Nom de l'Entreprise : LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications)

Adresse : Campus Scientifique, 615 Rue du Jardin-Botanique, 54506 Vandœuvre-lès-Nancy

Type d'activité (domaines couverts par l'entreprise) : Recherche en Informatique

Nom du parrain (enseignant IUT) : PARMENTIER Yannick

Mots-clés (sujets traités) : MQTT, VPN, Réseau, Raspberry Pi, Système

---

Résumé : Contribution à un projet de recherche existant : Conception d'une architecture de communication basée sur le protocole MQTT entre des capteurs connectés à des Rapsberry Pi. Utilisation de mosquitto (Broker MQTT), scripts bash, services systemd. Problématiques de charge sur le réseau, résilience, facilité d'utilisation et de mise en place.