ECOLE
**POLYTECHNIQUE**
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

# Deuxième ligne de titre du mémoire

## Ligne du sous-titre du mémoire

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en informatique à finalité spécialisée

**João Marques Correia**

Directeur
Professeur François Quitin

Co-Promoteur
Professeur Antoine Nonclercq

Superviseur
Quentin Delhaye

Service
BEAMS

Année académique
2019 - 2020

# Contents

# Chapter 1

# Introduction

## 1.1 Contexte du mémoire

## 1.2 Parties prenantes

## 1.3 Projet précédent

## 1.4 Structure du mémoire

# Chapter 2

# État de l'art

## 2.1 Technologies de communication à distance

Lors des dernières années, le nombre de dispositifs Internet des Objets (IoT) connaît une croissance fulgurante qui se maintiendra au cours des années qui suivent (voir figure 2.1). Ces objets sont capables d'acquérir des données sur leur environnement et/ou de prendre des actions sur celui-ci. Ils communiquent avec d'autres machines pour transmettre les informations acquises et recevoir des commandes lorsque cela est nécessaire. Les dispositifs IoT ont de diverses applications telles que les thermostats intelligents, les voitures connectées, suivi et monitorage d'actifs, et bien d'autres. Selon les conditions d'utilisation et son objectif, ces dispositifs doivent être capables d'envoyer des données sur des courtes ou des longues distances. En outre, leur consommation énergétique doit généralement rester faible, notamment lorsqu'ils sont alimentés par une batterie.

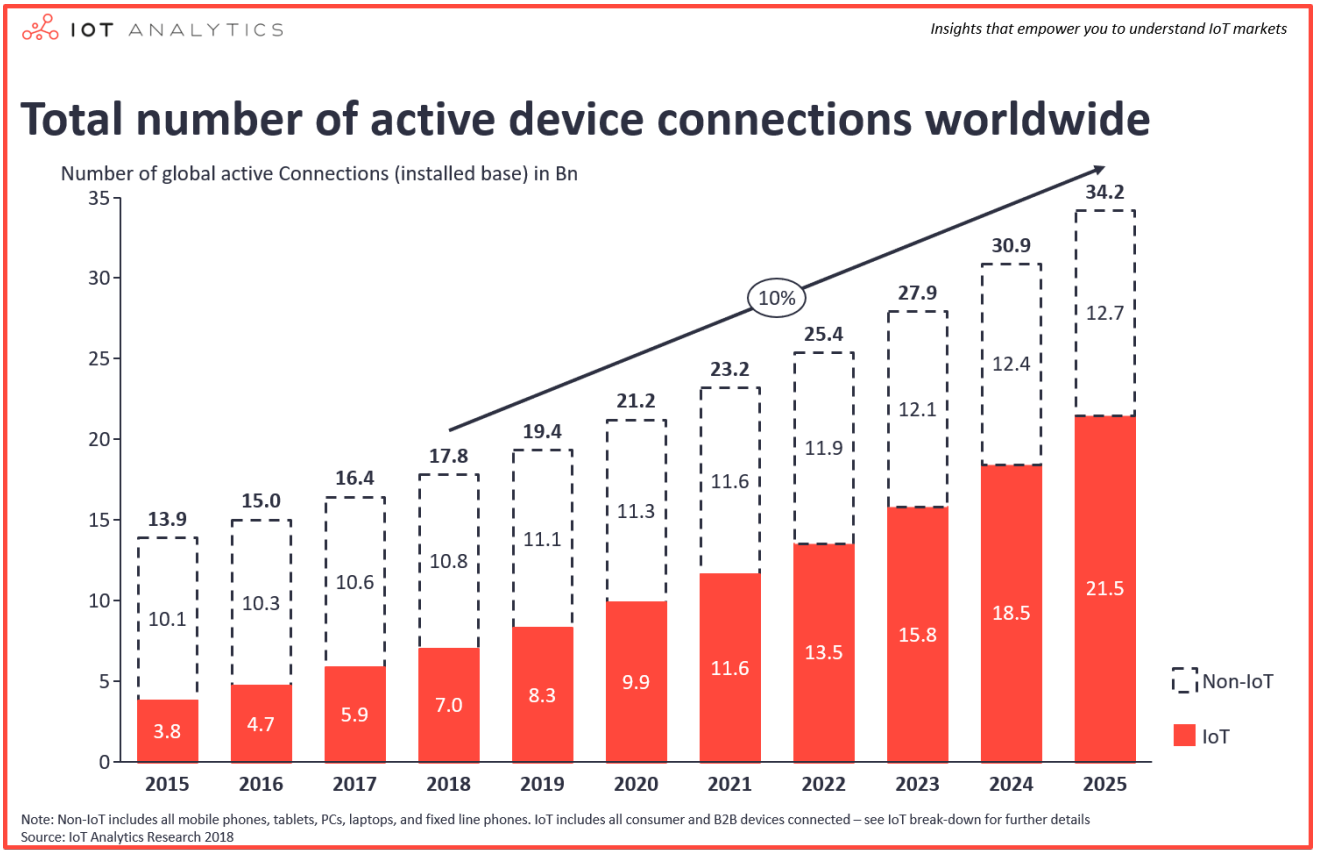


Figure 2.1: Nombre de dispositifs connectés à Internet [1]

Un réseau de communication unique et employable indépendamment du contexte du projet n'existe pas [2]. Toutefois, une multitude de réseaux avec des caractéristiques distinctes sont disponibles, tels que LoRaWAN et GSM. La figure 1 présente l'architecture simplifiée des réseaux utilisés par les dispositifs IoT pour échanger des informations avec des serveurs ou des utilisateurs qui sont connectés à Internet. Nous ne nous intéressons ici qu'aux technologies responsables de la transmission de données dans la phase 1 de la figure.

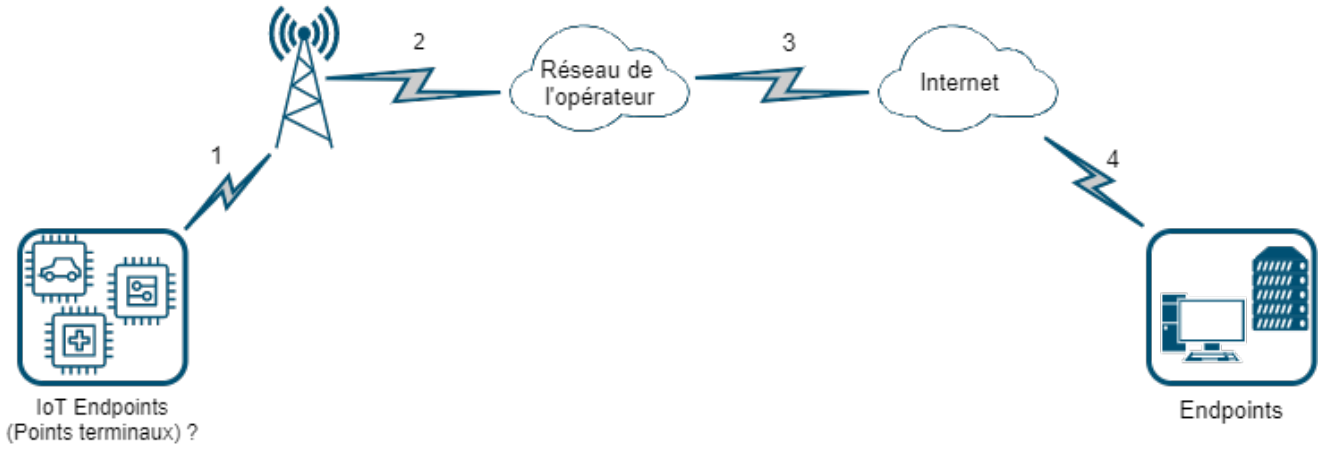


Figure 2.2: Architecture simplifiée d'un réseau avec dispositifs IoT (Basé sur [3, 4])

Actuellement, les principaux réseaux sans fil disponibles pour les dispositifs IoT sont :

- LoRaWAN
- Réseaux mobiles (2G/3G/4G/5G)
- Sigfox
- Satellite
- Wi-Fi
- Zigbee
- Bluetooth
- Réseaux mobiles IoT (NB-IoT et LTE-M)

Ces technologies utilisent toutes des ondes électromagnétiques pour transmettre les données, mais les fréquences sur lesquelles elles opèrent sont parfois très différentes. Certaines technologies utilisent une implémentation propriétaire, d'autres se basent sur des standards open source. [5] De façon similaire, certaines technologies exploitent la bande ISM[1], alors que d'autres exploitent des fréquences non réglementées. Ces dernières permettent de déployer son propre réseau privé, à condition d'acheter et configurer tout le matériel requis ce qui peut demander un investissement initial assez important. Ces différences accordent des propriétés distinctes aux réseaux en ce qui concerne le coût d'utilisation et déploiement, la bande passante, et la portée du signal. À

[1]Bande industrielle, scientifique et médicale

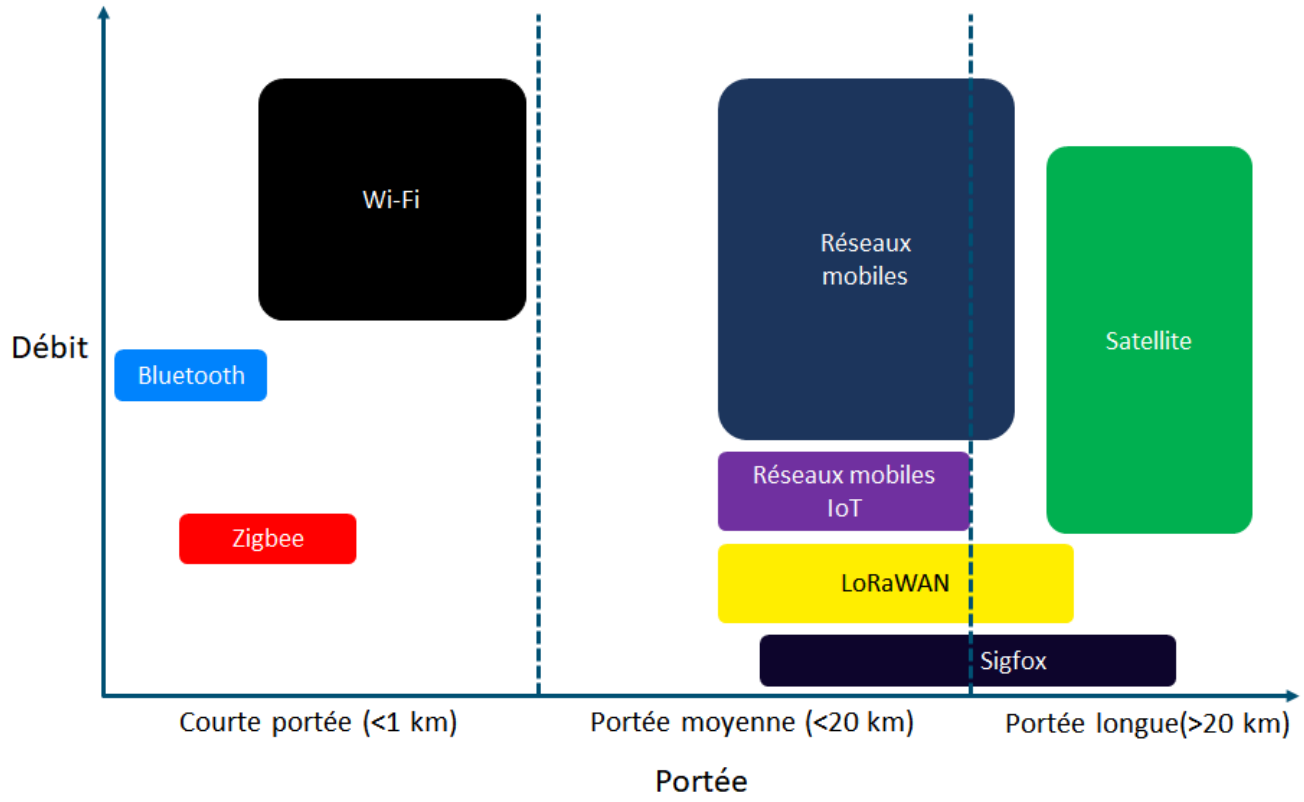


Figure 2.3: Classement des réseaux en fonction de leur portée et débit (Basé sur )

noter également, certains réseaux imposent une taille maximale sur chaque message, qu'il soit envoyé ou reçu. Le tableau comparatif de l'annexe A offre une vue d'ensemble sur les différentes technologies et leurs propriétés. Il faut cependant remarquer que la portée et le débit sont juste donnés à titre comparatif et ils sont à prendre avec des pincettes. En effet, ces valeurs varient beaucoup en fonction de certains paramètres tels que la géographie du milieu (urbain ou rural) et des éventuelles interférences. Néanmoins, le débit et la portée sont très importants pour classifier ces réseaux et, avec l'aide de la figure 2.3, ces technologies peuvent être classées de la façon suivante [6] :

- IoT haut débit : Dans cette catégorie se trouvent principalement les technologies comme les réseaux mobiles, le Wi-Fi et le satellite. Les débits proposés sont généralement supérieurs au Mb/s. En contrepartie, l'utilisation de ces technologies engendre une consommation énergétique plus élevée.
- IoT bas débit : Ici se trouvent les réseaux mobiles IoT, LoRaWAN, Sigfox, et Zigbee. En fonction du réseau, le débit peut varier entre quelques bits par seconde jusqu'à un Mb/s. Ces technologies offrent une consommation énergétique faible.
- IoT critique : C'est-à-dire lorsqu'il s'avère nécessaire de transmettre des données sur un intervalle de temps donné [6]. C'est une propriété des réseaux 5G.

Un point également important est que ces technologies ont de différents niveaux de maturité. Au moins un réseau mobile est disponible dans chaque pays, mais les nouveaux réseaux comme Sigfox, LoRaWAN, et 5G sont en revanche toujours en cours de déploiement. De ce fait, ces dernières technologies ne sont disponibles que dans certaines régions possédant une infrastructure moderne. La couverture du réseau est un critère important à prendre en compte avant de faire un choix

afin d'éviter les mauvaises surprises. Finalement, toujours dues à ce différent degré de maturité, certaines technologies plus anciennes pourraient voir la fin de ses jours dans les années à venir. Par exemple, certains opérateurs vont décommissionner leur réseau 2G à partir de 2020 [7]. Les opérateurs restants devraient suivre la même tendance dans les années suivantes. De plus, le réseau 3G devrait suivre la même tendance comme affichée sur la figure 2.4.

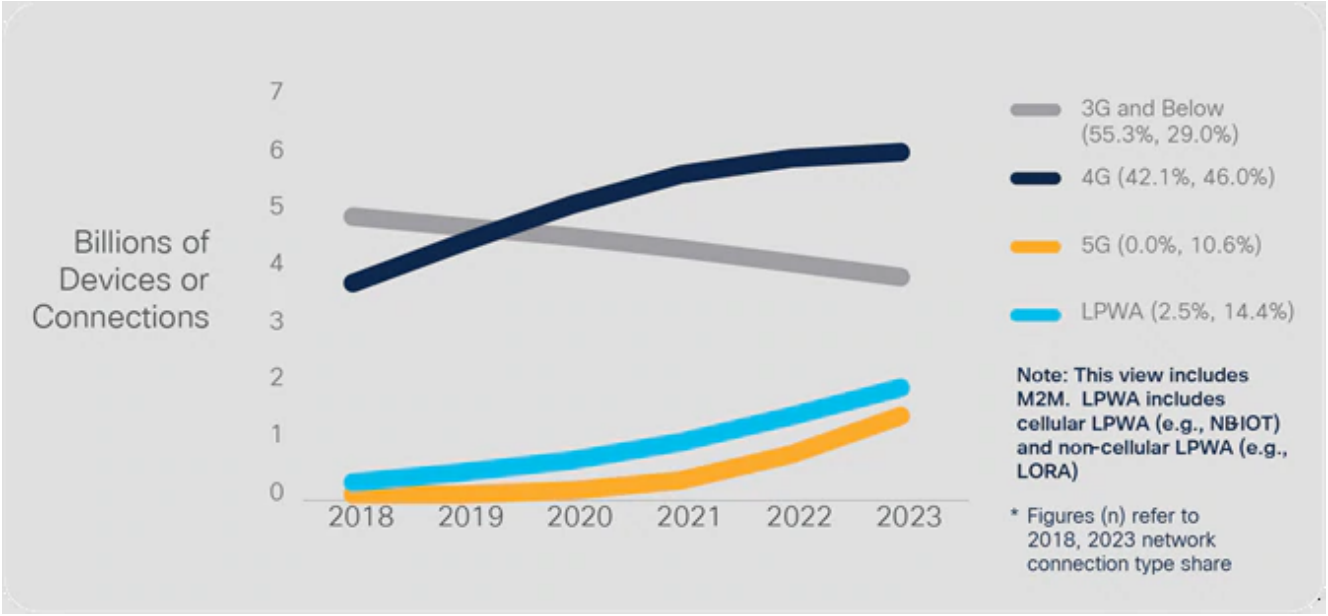


Figure 2.4: Évolution du nombre de dispositifs connectés aux différents réseaux [8]

## 2.2 Monitoring de l'état d'un serveur linux

La nécessité de surveiller toute l'infrastructure IT, et en particulier des serveurs tournant sous Linux, existe depuis un bon nombre d'années. Cependant, surveiller un système n'est pas trivial. Cela peut être accompli de différentes manières en fonction des évènements ou statistiques qui doivent être observées dans la machine hôte. Par exemple, l'utilisation de logs pour débugger une application ou voir l'historique de transactions d'une base de données est une technique de monitoring utilisée depuis de nombreuses années. D'autres techniques de monitoring existantes sont le profiling, le tracing et l'acquisition de métriques [9]. Dans cette section, seulement les technologies d'acquisition de métriques sont présentées puisqu'elles sont les plus utiles pour connaître l'état de la machine.

Créé à la fin des années 80, le protocole SNMP [10, 11] permet de monitorer et configurer des dispositifs différents connectés au réseau. Tout au long des années, des solutions de monitorage de métriques de plus haut niveau, ou tout-en-un, sont apparues sur le marché. Ces solutions utilisent différents protocoles, dont SNMP, pour surveiller les dispositifs sur le réseau et stockent toutes les informations recueillies sur une base de données. De plus, elles permettent de définir un ensemble de conditions à surveiller de plus près et si une de ces conditions est violée, une alerte est transmise vers l'équipe responsable de l'infrastructure. Deux exemples de solutions créées à la fin des années 90 et qui demeurent disponibles sont Nagios et Zabbix.

Actuellement, un grand nombre de solutions de monitoring sont disponibles sur le marché proposant différents modèles commerciaux. En effet, certaines solutions sont vendues comme un produit, mais d'autres préfèrent suivre un modèle open source. Dans le cas de ces dernières, les

entreprises responsables de l'application proposent souvent leurs services en matière d'assistance technique afin d'installer et maintenir la solution. Un autre modèle aussi appliqué est de proposer une version gratuite avec un nombre limité de fonctionnalités, et une version payante débloquant toutes les capacités de l'application. Avant d'entamer toute recherche d'une solution, il faut comprendre quelle formule s'adapte le mieux aux besoins de l'utilisateur. Comme il est affiché sur la figure 2.5, deux architectures existent pour les solutions de monitoring : *pull* et *push* [9, 12, 13, 12]. Chaque application est entièrement basée sur une de ces deux approches. Dans certains cas, une solution peut parfois supporter les deux architectures (voir exemple figure 2.6). Cependant, ces solutions détiennent toujours un modèle préféré, et l'autre possibilité sert juste à combler certaines lacunes du premier modèle puisque, en effet, chaque architecture possède ses avantages et faiblesses.

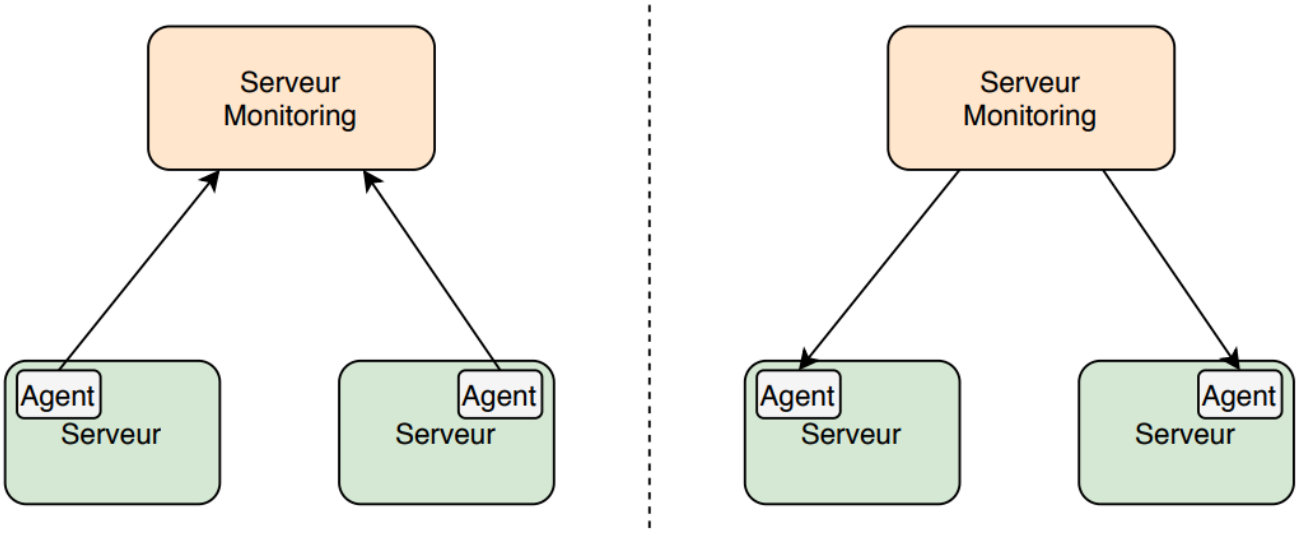


Figure 2.5: **Gauche :** Solution de monitoring avec architecture *push*. **Droite :** Solution de monitoring avec architecture *pull*.

Dans le modèle de type *push*, les agents dans la machine hôte envoient un ensemble de métriques ou des événements au serveur de monitoring. Cette technique permet notamment de définir des conditions d'alerte dans l'agent, et c'est celui-ci qui se charge de vérifier ces conditions. Lorsque le test des conditions passe, l'agent transmet un événement au serveur de monitoring. De plus, seul le système de push est capable de correctement acquérir des données à propos de jobs Batch de courte durée. [13, 14] En effet, comme affiché sur la figure 2.7, le modèle *pull* peut rater la fin du job ce qui se traduit par une perte d'informations.

Dans le modèle de type *pull*, les agents dans la machine hôte collectent et exposent les métriques, mais c'est au serveur de monitoring d'activement aller retrouver ces données. Par conséquent, les événements et alertes au niveau des agents ne sont pas pris en charge, dans la mesure où ces derniers ne savent pas quand le système de monitoring récupérera les informations. En revanche, cette méthode permet de mieux identifier si un problème s'est produit sur la machine hôte puisqu'elle ne répondra plus aux requêtes lorsque cela a lieu.[15] Ceci rend l'approche *pull* légèrement plus fiable que la technique *push*. Les deux méthodes possèdent donc leurs avantages et inconvénients qu'il faut tenir en compte lors du choix de la technologie. Cependant, dans la majorité des cas, les deux architectures offrent ces capacités très similaires [16].

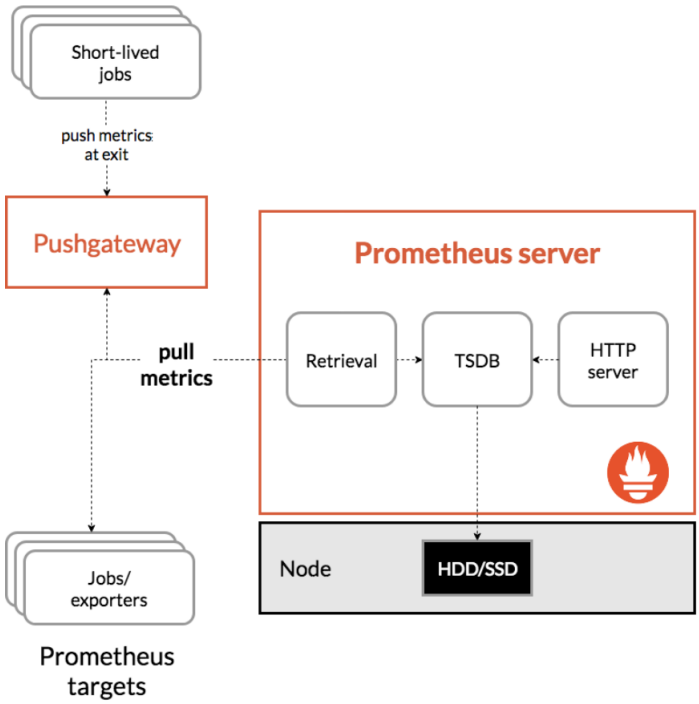


Figure 2.6: Architecture *pull* de Prometheus. Le modèle *push* est supporté grâce au Pushgateway.

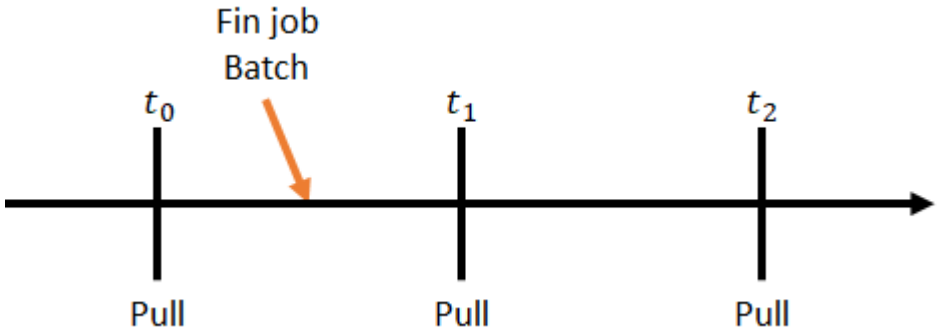


Figure 2.7: Faiblesse du modèle *pull*. Le système de monitoring a raté les informations de la fin du job

Indépendamment de l'architecture utilisée, toutes les informations recueillies ou reçues par le serveur de monitoring doivent être stockées sur une base de données. Les solutions plus anciennes comme Nagios et Zabbix, utilisent principalement des bases de données relationnelles [17, 18]. Ceci n'est plus le cas pour les solutions plus récentes telles que Prometheus et le TIG Stack[2] qui utilisent des bases de données orientées séries temporelles pour la persistance des données. Cette sorte de base de données, comme son nom indique, est mieux optimisée pour des valeurs horodatées [19]. En effet, ces bases de données possèdent les propriétés suivantes [20]:

- Elles supportent des écritures simultanées et avec de grands débits. Typiquement, dans ces bases de données se produisent énormément d'écritures, mais pas beaucoup d'accès.
- Ces bases de données doivent stocker d'énormes quantités de données temporelles. Elles utilisent des algorithmes de compression spécifiques pour stocker les données de manière efficace. [21]

[2]Le TIG stack est la combinaison de Telegraf, InfluxDB et Grafana pour créer un outil de monitoring

- Toutes les requêtes sont effectuées sur base d'un intervalle dans le temps
- Les données ont une durée de rétention. À partir d'une certaine date, la base de données élimine automatiquement les informations car elles sont jugées non utiles.

De ce fait, les bases de données orientées séries temporelles offrent un stockage plus efficace pour les métriques surveillées. Un dernier aspect très important d'une solution de monitoring est la visualisation de données. La majorité des solutions offrent soit un outil de visualisation propriétaire (Nagios ou Zabbix), soit elles exploitent des outils existants tels que Grafana (Prometheus et le TIG Stack).

## 2.3 Plateformes de visualisation de données

**Cette section n'est pas encore terminée.**

La visualisation de données consiste à représenter plusieurs formats de données sous un format visuel tel que les graphiques. Le système visuel humain est capable d'identifier des tendances ou des aberrations. Le but de la visualisation des données consiste alors dans l'aide de la compréhension des données en exploitant cette capacité. [22]

Les outils de visualisations de données sont alors des solutions software qui aident à accomplir cet objectif. [23] Ces outils fournissent une plateforme très puissante pour communiquer de l'information. Cependant, le concept de données est trop abstrait. En effet, elles peuvent être des coordonnées spatiales, les mesures de température d'une pièce, le nombre d'occurrences d'un évènement, le classement d'une course automobile, etc. Pour chacun de ces différents types de données, la méthode de visualisation à utiliser est différente. Par exemple, une carte serait utilisée pour afficher les coordonnées, mais une courbe serait préférée pour afficher l'évolution de la température. L'article [22] propose différentes méthodes et graphiques pour observer les différents types de données existants.

Comme présenté dans [source], la visualisation de données doit respecter les trois principes suivants : Thrustworthy, accessible, and elegant

Avec l'arrivée de l'ère du Big Data, les technologies de visualisation sont devenues d'autant plus importantes. En effet, la quantité de données est devenue tellement importante au point qu'un humain n'est plus capable d'analyser toutes les données. Ceci coïncide avec l'apparition de nouvelles technologies capables de mieux guider l'utilisateur à trouver des tendances dans les données. Actuellement, trois grandes catégories d'outils de visualisations de données sont disponibles :

- Les librairies software, telles que Matplotlib ou D3.js, qui aident les utilisateurs à créer de différentes visualisations de données. Ces outils demandent des compétences en programmation.
- Ensuite, il y a les outils purement pour la visualisation de données tels que Grafana ou Google Data Studio. À partir d'une interface web, ces outils permettent de facilement créer des graphiques en utilisant seulement des langages de requête comme SQL. Certains de ces outils sont capables d'auxilier l'utilisateur à formuler des requêtes. (Query Builders)

- Finalement, il y a les outils de Business Intelligence. Ils ont les mêmes fonctionnalités qu'un outil de visualisation de données. Cependant, ils permettent en plus de cela de faire un traitement des données et des prédictions sur celles-ci. Un exemple d'un tel outil est Tableau ou Power BI.

# Chapter 3

# Cahier des charges

Le but de ce mémoire consiste à développer un prototype capable de surveiller l'infrastructure de CERHIS et s'assurer du bon fonctionnement de celle-ci. De plus, ce système de monitoring doit pouvoir avertir un technicien lorsqu'un problème est détecté. Bien évidemment, pour accomplir cet objectif, le système devra acquérir des données sur le fonctionnement des différents dispositifs de CERHIS. Ces données devront être sauvegardées localement, mais aussi sur un serveur distant. Une interface web devra également être présente pour faciliter la visualisation de ces données, que cela soit dans le périmètre du centre hospitalier ou partout dans le monde. La figure 3.1 reprend l'architecture simplifiée de ce système de monitoring.

Pour simplifier la division du travail, la création du dispositif a été scindée en deux parties. Une première partie majoritairement *hardware* comportant le choix et l'assemblage de tous les composants nécessaires au développement du prototype. Cela englobe l'ordinateur requis pour tourner le logiciel, l'alimentation, la solution de communication et le software qui permet aux différents composants de s'interfacer. Ensuite, une deuxième partie comprenant juste le *software* nécessaire pour effectuer les tâches de monitoring. Cela comprend les fonctionnalités requises pour l'acquisition, traitement, stockage et visualisation des données. La présentation des spécificités de chaque partie se fera dans les sections suivantes (sections 3.1 et 3.2).

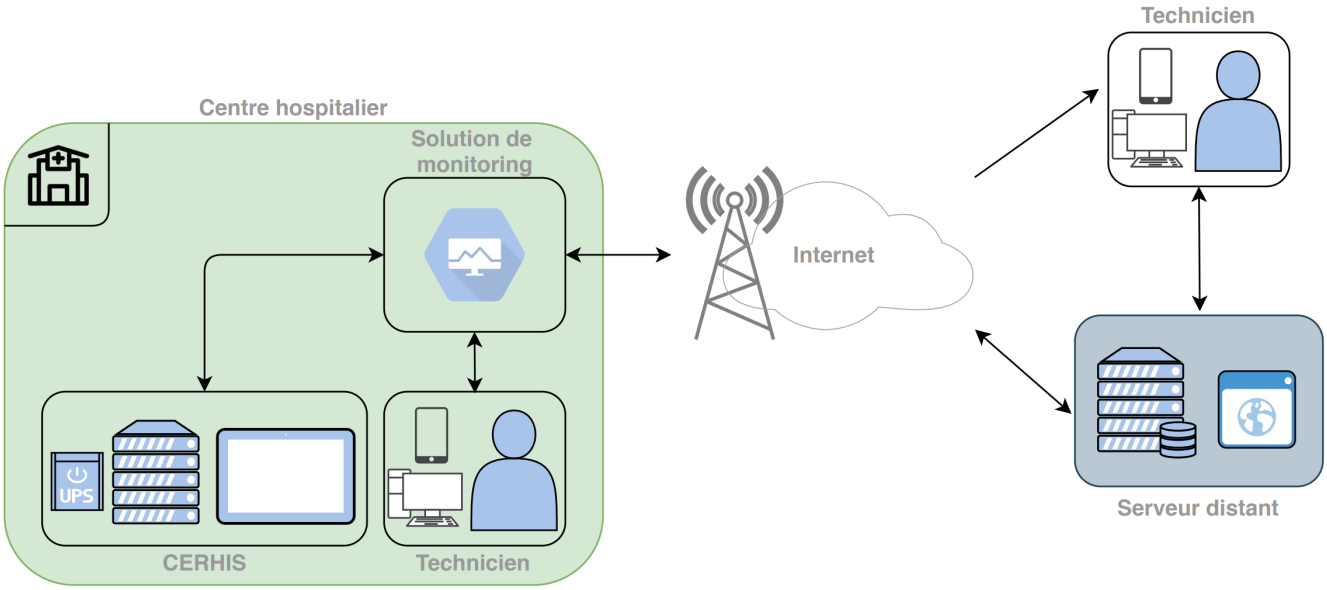


Figure 3.1: Architecture simplifiée de la solution de monitoring

## 3.1 Caractéristiques matérielles du dispositif

Étant donné que le dispositif de monitoring sera utilisé dans des centres hospitaliers en Afrique centrale, il doit être capable de résister à certaines caractéristiques propres du milieu telles que la température. Voici la liste exhaustive de toutes les caractéristiques matérielles que le produit doit posséder :

- Le prototype doit être robuste. Mes composants électroniques doivent notamment être capables de supporter des températures ambiantes supérieures à 30 °C pendant de longues périodes. [24]
- Le dispositif doit être capable de communiquer à distance, même lorsqu'une connexion à internet filaire n'est pas disponible. De ce fait, le prototype doit être capable de tirer parti d'une des technologies présentées dans la section 2 pour pouvoir envoyer des informations et des alertes.
- Le dispositif doit être d'apparence discrète, évitant d'attirer l'attention sur sa valeur.
- Le coût doit rester abordable et réaliste.
- Le dispositif doit posséder un système d'alimentation mixte capable d'alimenter le prototype pendant un ou deux jours en cas de passe du service électrique.
- Le boîtier doit être fermé hermétiquement pour empêcher des insectes d'y faire son nid.

## 3.2 Fonctionnalités à implémenter

Pour la section *software* du dispositif, les fonctionnalités à implémenter peuvent encore être divisées en deux sous-sections : la solution de monitoring tournant sur le site, et le serveur distant capable de recevoir, stocker et afficher des données. Les fonctionnalités présentées ci-dessous seront alors divisées en suivant ces deux sous-sections.

- **Solution de monitoring dans le centre hospitalier**
  - Effectuer un mappage du réseau local de façon à trouver tous les appareils qui y sont connectés.
  - Monitorer la présence des appareils connectés au réseau local. Si un appareil se déconnecte du réseau, pouvoir indiquer combien de temps est passé depuis sa dernière connexion.
  - Monitorer l'état de fonctionnement du serveur de CERHIS.
  - Envoi d'alertes par SMS.
  - Envoie de données vers un serveur distant.
  - Interface utilisateur permettant de facilement visualiser toutes les données.
  - Vérifier le niveau de la batterie de CERHIS et l'état de charge des tablettes.
  - Créer un historique de connexion des tablettes au serveur de CERHIS
- **Serveur distant**
  - Serveur capable de stocker les données envoyées depuis plusieurs centres hospitaliers.
  - Interface utilisateur permettant de facilement visualiser toutes les données disponibles sur le serveur distant.

## 3.3 Priorité des tâches et méthodologie de travail

Les listes des fonctionnalités et caractéristiques présentes ci-dessus reprennent tout ce qui serait nécessaire pour avoir un produit fini. L'objectif de ce mémoire est bien évidemment d'essayer d'implémenter le maximum de fonctionnalités possibles. Cependant, le temps est limité et des imprévus peuvent survenir au long du chemin. De ce fait, les diverses fonctionnalités ont été classées par ordre de priorité afin de garantir que le prototype rendu à la fin de ce mémoire possède au moins un certain nombre de fonctionnalités.

En premier lieu, c'était la sélection du matériel requis pour implémenter les fonctionnalités. Cela reprend, le choix de la plateforme, l'alimentation et la solution de communication. Cette tâche reprend aussi des tests sur la plateforme du projet de l'année précédente afin de vérifier si des changements étaient nécessaires.

Deuxièmement, c'était l'implémentation de tout le code nécessaire pour lier les divers composants hardware. En particulier, l'implémentation du code essentiel pour mettre en marche la solution de communication. Ici s'ajoutent aussi les premiers essais afin de prouver le bon fonctionnement de la plateforme.

Une fois la plateforme préliminaire choisie et testée, la possibilité de commencer à développer les différentes fonctionnalités du logiciel s'est ouverte. À nouveau, en raison du grand nombre de fonctionnalités requises, les plus cruciales ont été sélectionnées lors de réunions avec les parties prenantes. La liste ci-dessous présente les fonctionnalités triées par ordre décroissant de priorité :

- Monitorer l'état du serveur, en particulier surveiller l'uptime de celui-ci et avertir un technicien en cas de modification anormale de la valeur.
- Mapper le réseau local.
- Surveiller la présence des dispositifs de CERHIS.
- Monitorer l'état de certains processus dans le serveur.
- Envoi des données vers un serveur distant.
- Interface web pour la visualisation des données

Finalement, les dernières tâches reprennent la finalisation d'un boîtier permettant de loger les différents composants électroniques et des tests approfondis pour s'assurer du bon fonctionnement et la fiabilité de la solution de monitoring.

Pour essayer d'être le plus efficace possible, la méthodologie de travail consistait à écrire toutes les tâches liées à ce projet sur des notes Post-it. Les Post-its orange représentaient les missions liées au software, et les jaunes correspondaient aux missions liées au hardware. Chaque note possédait aussi un numéro coïncidant avec la priorité de la tâche. Le critère de sélection était en premier lieu la couleur, orange prioritaire sur le jaune, et ensuite la priorité (lorsque les Post-its avaient tous la même couleur).

# Chapter 4

# Prototype

## 4.1 Le premier prototype (2018-2019)

Dans le cadre du projet d'année de la première année du Master ingénieur civil en informatique, un prototype capable de surveiller l'infrastructure de CERHIS a été réalisé. Ce dispositif s'inspirait sur d'autres prototypes plus anciens créés par des étudiants de l'École polytechnique de Bruxelles. L'architecture du dispositif produit l'année dernière est présentée sur la figure 4.1.

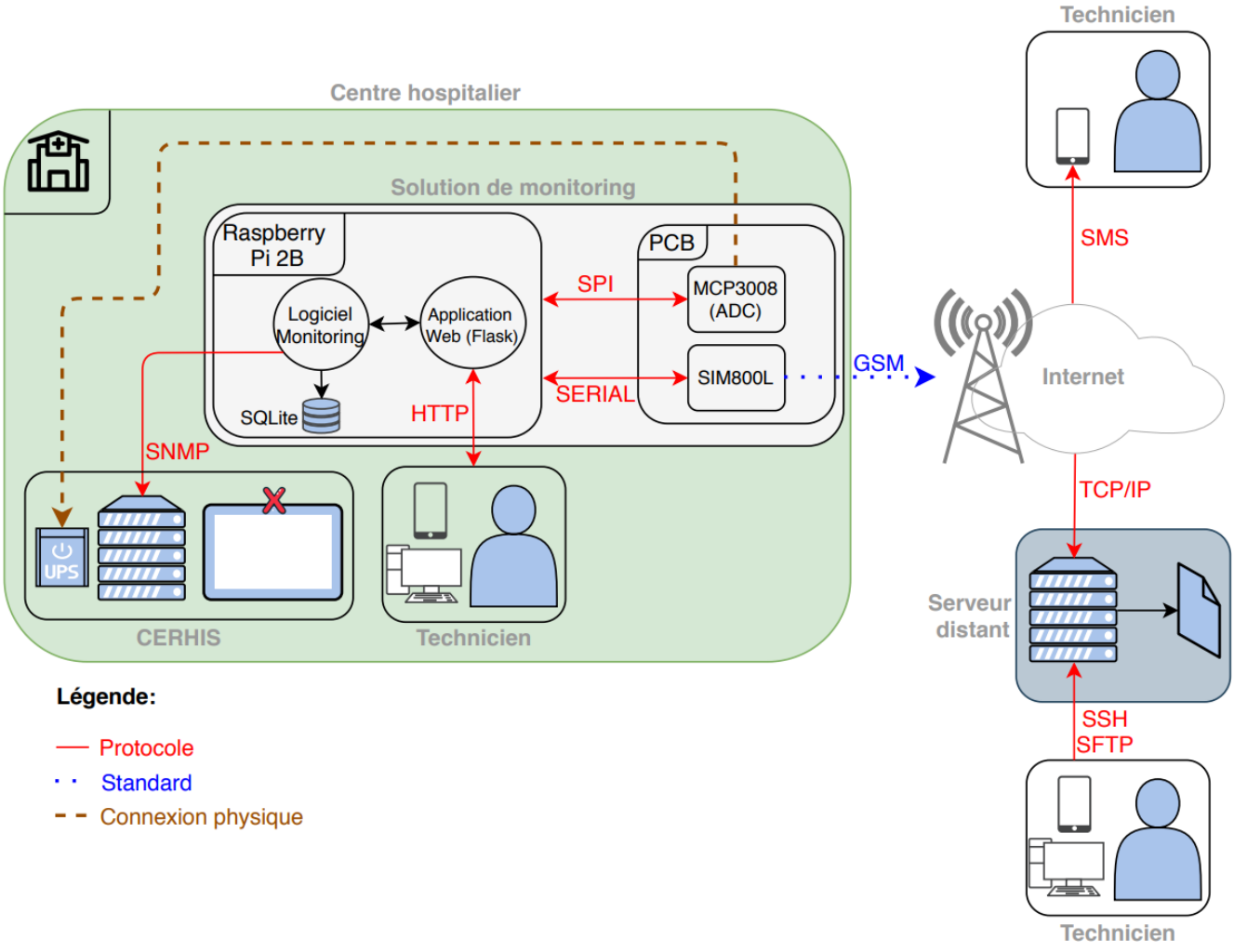


Figure 4.1: Architecture de la solution de monitoring développée lors de l'année académique 2018-2019

Un Raspberry Pi 2B était utilisé pour tourner le logiciel de monitoring ainsi que l'interface web permettant la configuration de celui-ci. Un circuit imprimé (PCB) avait été réalisé dans le but de loger des composants électroniques différents tels que le module de communication GSM SIM800L et le convertisseur analogique numérique MCP3008. Ce même circuit imprimé s'insérait sur les ports GPIO du Raspberry Pi afin que ce dernier puisse interfacer avec les différents composants présents sur le PCB. Grâce au SIM800L, le réseau GSM d'un opérateur mobile local pouvait être utilisé pour envoyer des SMS vers un technicien, ou des informations quelques conques vers un serveur distant. Dans ce premier essai, ce serveur était juste capable de recevoir les données envoyées par le prototype et de les stocker sur un fichier *.txt*. Aucun traitement ou visualisation des données n'était offert aux techniciens. De plus, le monitoring des tablettes n'a pas pu être implémenté par manque de temps.

De façon à alimenter le dispositif, un système d'alimentation mixte avait été mis en place. Comme le montre la figure 4.2, ce système était basé sur une batterie acide-plomb de 12V, un régulateur de charge permettant de recharger la batterie et un convertisseur step-down transformant la tension de 12 à 5 volts.

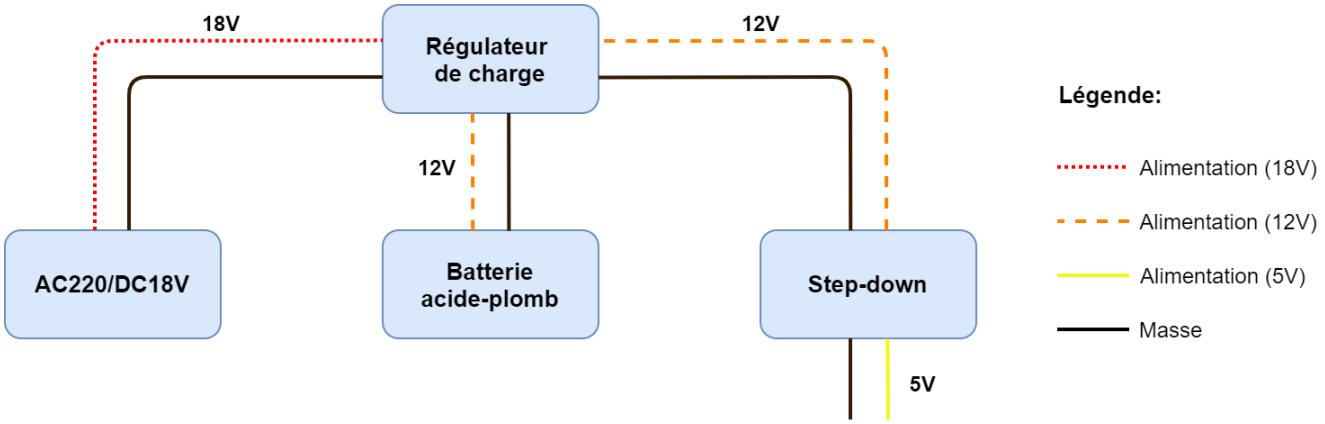


Figure 4.2: Système d'alimentation de la solution de monitoring développée lors de l'année académique 2018-2019

Ce prototype a été testé pendant plusieurs semaines[1] à Goma par un étudiant de l'ISIG[2]. Lors de cette période de tests, plusieurs problèmes ont été identifiés. Ils seront présentés dans la section suivante.

### 4.1.1 Problèmes détectés

Plusieurs protocoles de tests avaient été mis en place l'année dernière dans le but d'évaluer ce premier dispositif. Les soucis qui ont été trouvés lors de ces tests peuvent être classés selon trois grandes catégories : communication, alimentation et température. Tous les problèmes sont documentés ci-dessous.

- **Communication :**
  - Corruption de certaines données échangées entre le SIM800L et le Raspberry Pi. Des bits semblaient être modifiés lors de la transmission des messages entre les deux composants. Cela provoquait une perte d'informations. De plus, cela rendait plus difficile

[1] entre le 14 mars et le 10 avril 2019
[2] Institut supérieur d'informatique et de gestion de Goma

l'utilisation du SIM800L, car ce dernier n'était pas capable de traiter toutes les commandes qui lui étaient envoyées.

  - Implémenter toutes les fonctionnalités de communication requises avec le SIM800L était très laborieux. En effet, ce module peut être vu comme un automate fini déterministe. Des commandes AT sont utilisées pour contrôler le module, c'est-à-dire qu'elles sont employées pour changer l'état de celui-ci. Une succession correcte de multiples commandes est nécessaire afin d'accomplir une simple tâche telle que l'envoi d'un SMS. Ceci rendait le procès d'implémentation assez long, en particulier l'implémentation de la gestion des erreurs. Le problème mentionné ci-dessus a encore plus aggravé la complexité de la manipulation de ce module.

- **Alimentation :**

  - Le régulateur de charge initialement utilisé est tombé en panne lors de la réalisation des tests. Un simple remplacement de cette pièce semble avoir résolu le problème.
  - Le step-down est tombé en panne lors des tests. Malheureusement, aucun remplacement pour cette pièce a été trouvé à Goma. Ce souci est possiblement lié au problème de température qui sera présenté ci-dessous. Cependant, n'ayant pas assez d'information pour prouver l'existence d'un lien de causalité, chaque problème doit être considéré séparément.

- **Température :**

  - D'après les personnes en charge de tester le prototype, celui-ci semblait atteindre de hautes températures. Des températures élevées peuvent avoir un impact considérable sur la durée de vie des différents composants électroniques.

## 4.2 Nouveau Prototype

Compte tenu des problèmes rencontrés l'année précédente, un nouveau prototype a dû être conçu. Toutefois, énormément d'informations ont été acquises au cours de l'année dernière. Il ne serait pas judicieux de partir de zéro, car cela demanderait un effort assez important pour atteindre le niveau précédent. Ce nouveau prototype correspond alors à une évolution du résultat de l'année antérieure, ayant comme but de combler toutes les lacunes de ce dernier. Le choix de remplacer chaque composant est abordé ci-dessous.

### 4.2.1 Solutions considérées

#### Choix de la plateforme

Lors de l'année précédente, un Raspberry Pi 2B a été utilisé en tant que plateforme principale du projet. Ce modèle en particulier avait été choisi en raison de sa consommation énergétique plus faible que celle des modèles suivants, tout en offrant des performances amplement suffisantes. Depuis le projet antérieur, une nouvelle version est disponible sur le marché, le Raspberry Pi 4. Cependant, ce dernier présente une consommation énergétique encore plus élevée sans offrir de nouveaux avantages au projet. De ce fait, puisque le Raspberry Pi 2B s'est montré très fiable et qu'il restera en production jusqu'en 2026 [25], il sera à nouveau utilisé dans ce nouveau prototype. Dans l'annexe (ajouter_annexe), le tableau comparatif de toutes les solutions considérées l'année dernière est présent à titre de référence.

### Solution de communication

La communication à distance est probablement un des éléments les plus complexes de ce projet. En effet, l'infrastructure des différents réseaux en Afrique Centrale est sous-développée comparée à l'infrastructure en Europe. De plus, la partie de la communication est celle qui a apporté le plus de problèmes l'année précédente, et celle dont les problèmes sont les plus difficiles à résoudre. De ce fait, cette partie doit être abordée avec caution et la décision doit être très pondérée.

Actuellement, le choix de réseaux accessibles en Afrique centrale est très restreint. En particulier, les réseaux orientés IoT (LTE-M, NB-IoT, Sigfox, LoRaWAN) ne sont actuellement pas disponibles dans la majorité des pays en Afrique Centrale ou leur couverture est extrêmement limitée. De plus, les réseaux Sigfox et LoRaWAN[3] sont très contraignants sur la quantité de données qui peut être envoyée sur une journée. De ce fait, ils ne s'encadrent pas dans le cahier des charges de ce projet. Le satellite aurait été le choix incontestable si le coût n'était pas un facteur limitant. Les options restantes telles que le Wi-Fi et le Bluetooth ont une portée très courte. Les réseaux mobiles se présentent donc comme étant le meilleur compromis entre disponibilité, coût et consommation énergétique. L'accès à ces derniers est relativement facile, et la couverture des réseaux 2G et 3G semble être relativement bonne dans les régions plus densement peuplées. [26, 27]

Toutefois, le développement des réseaux mobiles orientés IoT (NB-IoT et LTE-M) dans ces pays doit être suivi de près au cours des prochaines années. Ils s'encadrent dans l'esprit de ce projet et offrent potentiellement un meilleur compromis que les réseaux mobiles classiques en raison de leur plus grande portée et de leur faible consommation énergétique. Un autre aspect important à surveiller est la suppression des réseaux 2G au cours des années qui suivent. Cependant, vu le retard de l'infrastructure dans certains pays d'Afrique Centrale, il est très peu probable que cela ait lieu prochainement.

Les réseaux mobiles proposent plusieurs protocoles pour transmettre des données. Afin de choisir le protocole le plus approprié pour ce projet, il est d'abord nécessaire de comprendre exactement quels types de communication sont requis. En premier lieu, il faut considérer la communication de machine à personne. Le prototype doit être capable d'envoyer un message d'alerte à un technicien. Le cahier des charges impose que ce type de communication soit effectué par SMS. Les techniciens ayant tous accès à un téléphone, cette méthode semble être la plus efficace pour communiquer avec eux.

Ensuite, il y a la communication de machine à machine (M2M). Le prototype doit être capable d'envoyer les données recueillies vers un serveur distant qui les rendra accessibles aux techniciens. Cet échange de données peut être effectué également par SMS. Cependant, comme démontré dans le rapport de l'année dernière, le SMS est une méthode très coûteuse pour l'envoi de données M2M. D'autres protocoles permettent de transporter des données à un moindre coût comme l'USSD et le GPRS/HSPA/LTE. De ce fait, une comparaison a été effectuée afin de comprendre les avantages de chaque protocole, ainsi que celui qui est le plus approprié pour ce prototype.

L'USSD, ou Unstructured Supplementary Service Data, est un protocole de communication présent dans les réseaux mobiles (GSM, 3G, 4G) qui permet à un client de communiquer avec les serveurs de l'opérateur du réseau mobile. L'USSD est un stateful protocol et les échanges de données entre le client et le serveur se font en temps réel. Les opérateurs utilisent très souvent ce protocole pour donner accès à certains services spéciaux, tels que vérifier le solde restant dans une carte prépayée

[3]Les contraintes sur la quantité de données pour les réseaux LoRaWAN ne sont applicables que sur les réseaux publics.

($XXX\#$). L'USSD présente la caractéristique intéressante que toutes les connexions initiées par le client sont toujours routées vers les serveurs de l'opérateur de la carte SIM, même lorsque le client se trouve en itinérance [28]. Cela garantit que, indépendamment du lieu où se trouve le client dans le monde, l'échange d'informations se déroule toujours de manière similaire. De plus, cela permet aussi de diminuer les coûts élevés souvent associés à l'itinérance. Mais le protocole USSD possède bien d'autres avantages tels que :

- Aucune configuration supplémentaire n'est requise pour utiliser l'USSD dans le réseau d'un opérateur. Au contraire, l'Access Point Name doit être configuré avant d'avoir accès au service GPRS.
- Le protocole USSD utilise moins de ressources d'un modem ou du réseau d'un opérateur que le GPRS. [29]
- Le protocole USSD reste disponible même lorsque la connectivité à Internet du réseau est coupée. En contrepartie, cela rendrait le service GPRS inexploitable.[30]
- La majorité des modems sur le marché, dont le SIM800L, sont compatibles avec USSD. [28]

Cependant, le protocole USSD possède aussi les désavantages suivants :

- Déployer une application utilisant USSD est très difficile puisque cela demande de travailler de très près avec un opérateur de télécommunications. En effet, toutes les connexions sont dirigées vers les serveurs de ces derniers. [31]
- Lié au problème précédent, le protocole USSD ne donne pas directement accès à Internet. De ce fait, il n'est par exemple pas trivial d'utiliser un protocole application tel que HTTP pour envoyer des données vers un serveur.

Pour pallier ce dernier problème, une solution assez particulière est exploitée actuellement dans certains pays en Afrique [32]. Elle permet aux utilisateurs d'accéder à des services qui ne sont disponibles que sur Internet, mais à travers le protocole USSD. La figure 4.3 illustre comment fonctionne cette solution. Le serveur USSD joue le rôle de proxy et convertit les messages USSD initialement envoyés par le client en requêtes HTTP à envoyer à un serveur distant. Malheureusement, cette solution n'est disponible que dans un nombre restreint de pays, mais une solution similaire pourrait être éventuellement utilisée pour envoyer les données au serveur distant.

L'alternative à l'USSD est de recourir aux normes General Packet Radio Service (GPRS), High Speed Packet Access (HSPA) ou Long Term Evolution (LTE) pour transmettre des données. Ces normes permettent de communiquer des données à travers le réseau de l'opérateur et fournissent au client un accès à Internet. Les protocoles de la couche application tels que HTTP sont très souvent utilisés en conjonction avec ces normes. Un exemple est l'emploi d'un smartphone connecté à Internet par le biais de données mobiles pour accéder à un site web. Ce niveau d'abstraction supplémentaire facilite l'utilisation de cette solution.

Trancher entre l'USSD et le GPRS n'est pas simple. D'une part, l'USSD semble offrir une solution très fiable qui est couramment employée dans plusieurs pays africains. De l'autre côté, le GPRS permettrait d'exploiter les avantages d'une connexion standard à Internet et tous les protocoles d'application qui viennent avec. De ce fait, une comparaison entre les possibles de chaque solution a été effectuée.

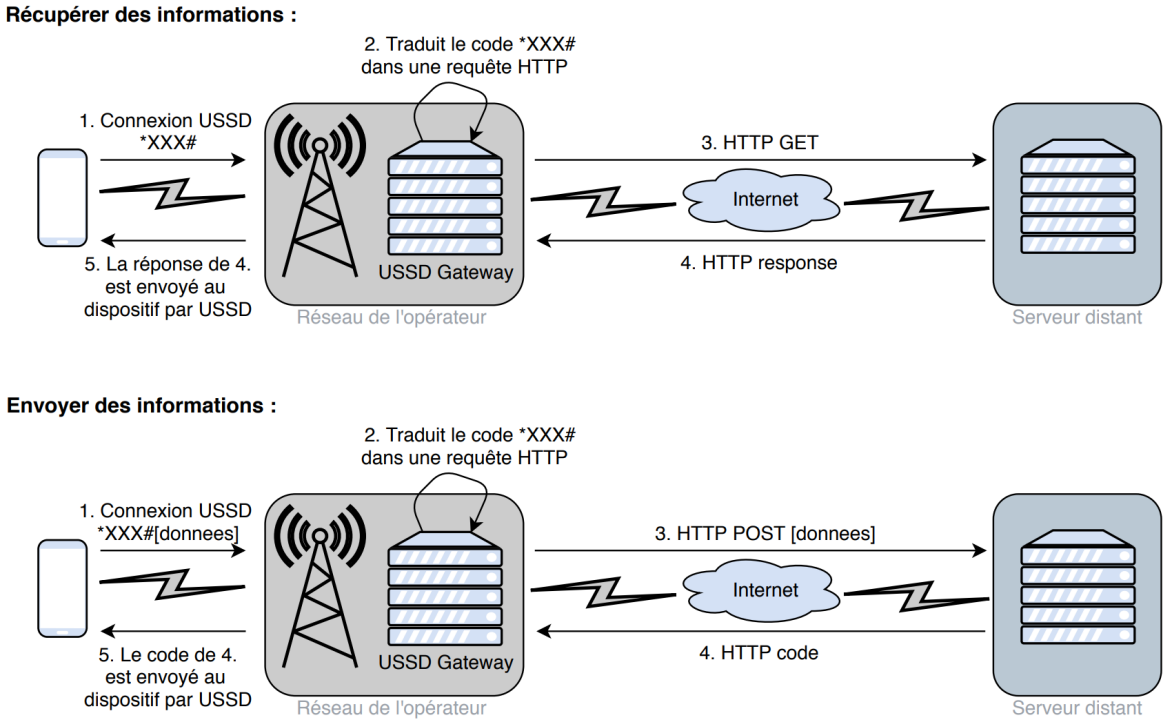


Figure 4.3: Architecture simplifiée d'une solution USSD permettant d'accéder à des services sur Internet

### Solution 1: Routeur LTE (GPRS/HSPA/LTE)

Dans cette première implémentation, l'idée était de remplacer le module SIM800L par un routeur LTE. Ce type de routeur est capable de se connecter aux réseaux mobiles afin de permettre aux clients d'accéder à Internet. Toute la partie bas niveau des réseaux serait alors traitée par le Raspberry Pi et le routeur. De ce fait, le logiciel de monitoring n'aurait besoin que d'utiliser des protocoles de la couche application, simplifiant donc l'implémentation du logiciel. Certains de ces routeurs peuvent aussi être raccordés à un deuxième réseau WAN[4], comme une potentielle connexion Internet du centre hospitalier. Cette solution est la seule à proposer ce type de redondance. Cependant, les routeurs 4G sont assez coûteux, principalement ceux destinés à des environnements industriels. Un investissement d'au moins 70 ou 80 euros serait requis pour acquérir tel dispositif. De plus, leur consommation énergétique est assez élevée comparée au SIM800L puisque les routeurs proposent un grand nombre de fonctionnalités telles que le Wi-Fi et plusieurs ports Ethernet. La figure 4.4 illustre l'architecture simplifiée de cette solution.

### Solution 2: Service USSD avec SIM800L

Jusqu'à il y a quelques années, déployer une application USSD était assez complexe puisque l'aide d'un opérateur était nécessaire [31]. Toutefois, sont apparues récemment sur le marché des entreprises qui offrent des services de communication exploitant la technologie USSD. En fonction de l'entreprise en question, la couverture du service ainsi que ses fonctionnalités diffèrent. Utiliser une telle solution permet de très aisément exploiter les avantages du protocole USSD tout en mitigeant ses défauts. Thingstream est un service de communication qui exploite la technologie USSD pour permettre des dispositifs de communiquer entre eux presque partout dans le monde. Cette plateforme se distingue par le fait qu'elle propose le protocole MQTT sur le protocole USSD. De ce fait, comme pour la solution précédente, le logiciel de monitoring a accès à un protocole de la couche application ce qui facilite l'échange de données, mais son choix est imposé. La solution

[4]Wide Area Network ou réseau étendu en français

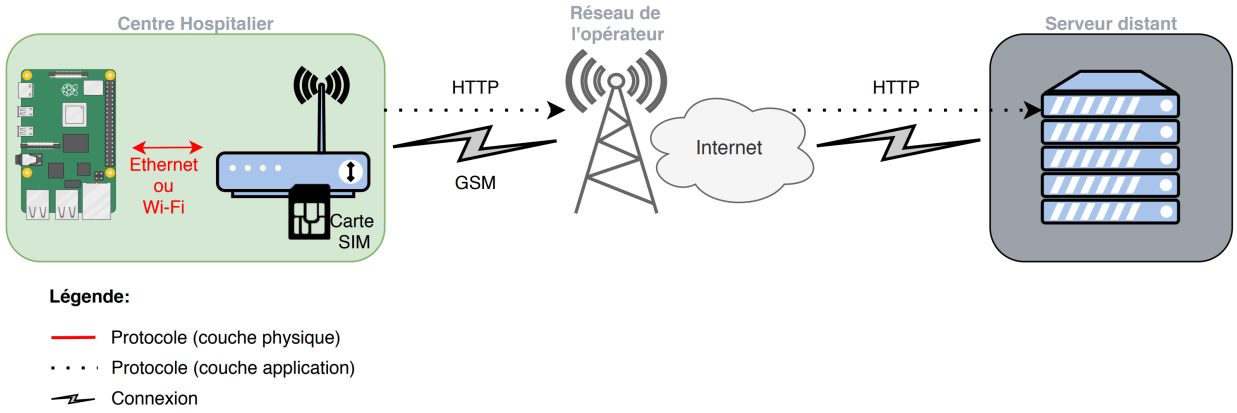


Figure 4.4: Architecture simplifiée de la solution de communication avec un router LTE

de Thinsgtream sera présentée plus en détail dans la section 4.2.2.

### Solution 3: SIM800L et protocole point à point

Dans cette implémentation, le SIM800L est à nouveau exploité. En effet, l'architecture est très similaire à celle du prototype de l'année dernière, la différence réside dans l'emploi d'un protocole de plus haut niveau pour communiquer avec le modem. L'année précédente, les commandes AT étaient envoyées par le logiciel de monitoring au SIM800L à travers le protocole SERIAL. Pour simplifier ce processus, il est possible d'utiliser le protocole point à point (PPP). Ce protocole de la couche liaison de données gère toute la complexité des commandes AT et permet au Raspberry Pi de se connecter à Internet. Le logiciel de monitoring peut donc de recourir aux protocoles habituels de la couche application pour échanger les données avec le serveur distant. Veuillez noter que le protocole SERIAL est toujours utilisé dans la couche physique. La figure 4.5 illustre le fonctionnement de cette solution.

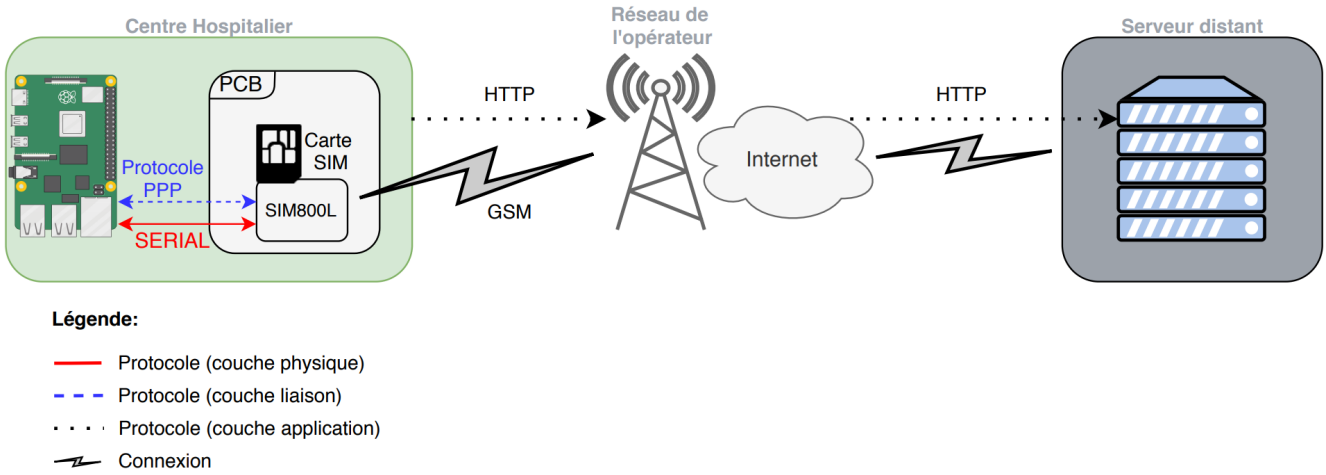


Figure 4.5: Architecture simplifiée de la solution de communication avec le SIM800L et le protocole PPP

Finalement, la solution 2 (USSD) est celle qui a été implémentée lors de ce mémoire. La solution 1 demandait un investissement assez élevé. De plus, il n'y a pas énormément d'informations sur la qualité des réseaux en République Démocratique du Congo [33]. De ce fait, utiliser une solution basée sur le protocole USSD est le choix le plus sûr puisque, théoriquement, ce protocole fonctionne même lorsque la connectivité Internet du réseau est indisponible.

Veuillez noter que la troisième solution ne possède pas le même problème de coût que la première. Cependant, cette solution a seulement été identifiée quelques semaines plus tard que les deux autres et, à ce stade, la deuxième méthode avait déjà commencé à être implémentée. En outre, la troisième proposition reste théoriquement moins fiable qu'une solution basée sur USSD. De ce fait, il a été décidé de continuer l'implémentation de la deuxième méthode afin d'augmenter la probabilité d'avoir une solution complètement fonctionnelle à la fin de ce mémoire. Utiliser le temps disponible pour implémenter les deux solutions parallèlement pourrait conduire à des solutions sous-optimales ou non fonctionnelles.

### Alimentation

Le système d'alimentation créé l'année dernière a aussi présenté des problèmes. En effet, le régulateur de charge ainsi que le step-down sont tombés en panne. Cependant, l'idée semblait prometteuse et, en particulier, l'autonomie du système était très bonne. Par conséquent, le plan pour ce mémoire fut de poursuivre avec la même idée, mais de changer deux choses. Premièrement, le step-down précédent a été remplacé par un nouveau modèle mieux documenté. Ce nouveau step-down est le XL4015.

Le XL4015 est un convertisseur Buck capable de supporter une tension d'entrée comprise entre 8V et 36V et des charges allant jusqu'à 5A. Le rendement de ce convertisseur est très élevé pouvant atteindre les 96%. Des applications typiques pour ce step-down sont des moniteurs LCD et des appareils télécom ou réseaux. Cette dernière application s'encadre dans ce projet puisqu'un modem est inclus dans le prototype.

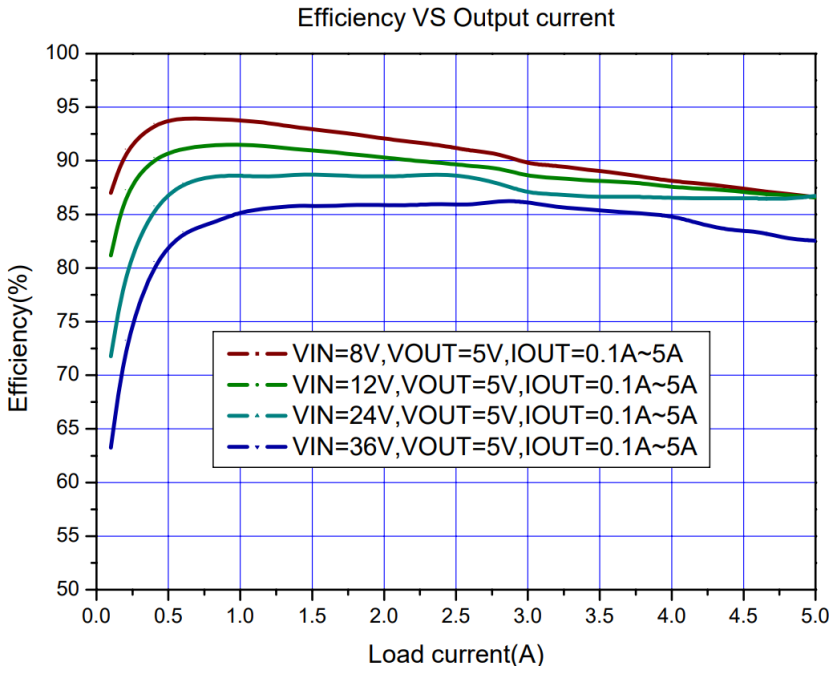


Figure 4.6: Courbe de rendement du XL4015 lorsque $V_{out} = 5V$ [34]

Ensuite, l'approche pour tester le système d'alimentation a été modifiée. Dans un premier temps, le prototype a été seulement alimenté à partir d'un chargeur de 12V. La batterie et le régulateur de charge n'étaient alors pas utilisés. Si le temps le permettait, le prototype serait testé avec la solution d'alimentation complète lors d'une deuxième phase. Ces tests déphasés ont comme but de mieux aider à identifier le composant provoquant les problèmes

**Boîtier**

Une fois les composants définis, il ne restait plus qu'à choisir le boîtier. L'importance de celui-ci ne doit pas être sous-estimée. En effet, le problème de température du prototype précédent était probablement lié à la mauvaise ventilation de la boîte. De plus, le boîtier utilisé précédemment n'était pas adapté pour loger les composants électroniques différents. Les modifications requises pour l'ajuster aux composants n'étaient pas simples à effectuer et l'accès aux divers ports I/O du Raspberry Pi était très limité. Cependant, avant de partir sur la conception de la nouvelle boîte, plusieurs tests ont été réalisés sur l'ancien boîtier afin d'examiner les problèmes de température et la nécessité d'ajouter un ventilateur. Les résultats de ces tests seront présentés dans la section 4.2.3 et le nouveau design sera abordé dans la section 4.2.4.

### 4.2.2 Thingstream

Thingstream est une entreprise qui propose *IoT-Communication-as-a-Service*. En d'autres termes, ils fournissent une plateforme qui facilite la communication entre des dispositifs différents. Leur service est disponible dans plus de 190 pays grâce à l'utilisation des réseaux mobiles GSM des opérateurs de télécommunication locaux. En particulier, le service est capable de basculer entre les réseaux des différents opérateurs disponibles à un endroit donné. Cette redondance accorde une grande fiabilité au service.

La grande particularité de Thingstream est que l'ensemble de la plateforme est basé sur le protocole MQTT pour la transmission des messages. Ce protocole a été spécifiquement conçu pour des "réseaux à faible bande passante, à haute latence et non fiables"[35]. Un SDK[5] est fourni afin de simplifier l'accès à la plateforme de Thinsgtream et d'exploiter le protocole MQTT. Actuellement, Thingstream ne possède pas de concurrents directs offrant les mêmes services. Des services similaires se concentrent souvent uniquement sur la proposition d'une connectivité Internet à travers les réseaux mobiles. Un exemple d'un tel service est EMnify. Cependant, aucun software est proposé pour aider à plus facilement exploiter la plateforme. Le niveau d'abstraction supplémentaire fourni par Thingstream est une plus-value très importante, particulièrement dans le cadre de ce mémoire où le temps est limité.

L'architecture de la plateforme de Thingstream est présentée sur la figure 4.7. Les SN-Things correspondent aux appareils qui se connectent au réseau de Thingstream par le biais de réseaux mobiles. Comme indiqué sur la figure, ces dispositifs utilisent le protocole USSD pour transporter les messages de la couche application. La couche application est gérée par le protocole MQTT-SN. Ce protocole est une variante du protocole MQTT qui a été spécifiquement conçue pour les Sensor Networks. Le MQTT-SN permet notamment aux applications qui ne peuvent pas utiliser les réseaux TCP/IP de communiquer avec des applications MQTT. La spécification complète de MQTT-SN est disponible sur [36].

Ensuite, il y a les IP-Things. Ils correspondent aux différents clients qui se connectent au broker[6] en utilisant le protocole MQTT et qui ont donc accès à un réseau TCP/IP. Grâce à une fonctionnalité nommée Data Flow, Thingstream permet aussi au broker de communiquer avec d'autres applications en utilisant d'autres protocoles que MQTT. Actuellement, les protocoles disponibles sont :

- SMS

[5]Software Development Kit ou kit de développement logiciel

[6]Le broker est un serveur qui reçoit tous les messages publiés et les distribue vers les clients appropriés.

- HTTP
- FTP
- SMTP (email)

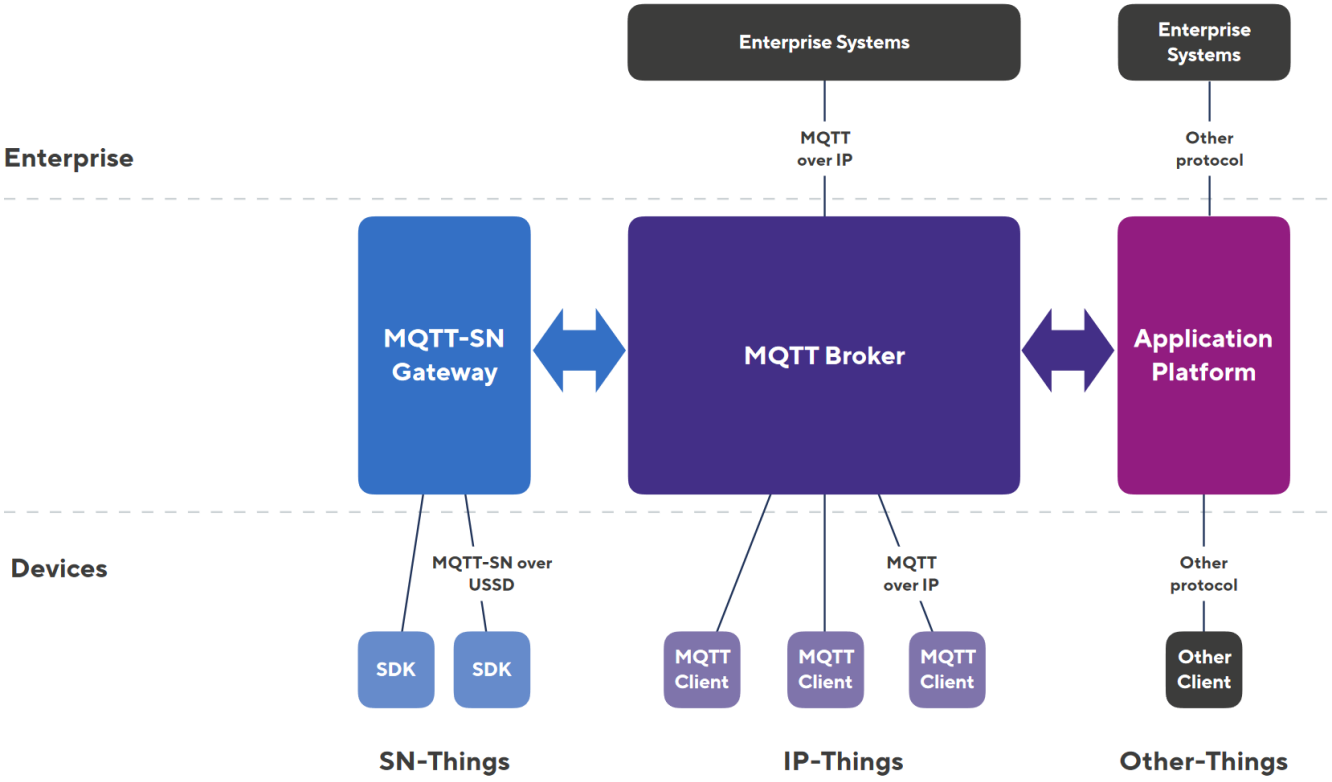


Figure 4.7: Architecture simplifiée de la plateforme Thingstream [37]

Plus de détails sur cette fonctionnalité sont disponible sur [38]. Cependant, il est très important de noter que Thingstream offre la possibilité d'envoyer des SMS. De ce fait, les deux types de communication requis pour ce projet sont supportés par Thingstream.

En résumé, Thingstream offre un service exploitant tous les avantages du protocole USSD, mais sans devoir passer directement par un opérateur. Cependant, l'utilisation de ce service n'est pas sans risque. Toute la partie de communication du projet sera dépendante de cette tierce partie. Par conséquent, si Thingstream, pour une raison quelconque, cesse d'offrir ce service, la partie communication du prototype sera complètement inopérante. De façon à limiter le possible impact de ce risque, des mesures ont été prises lors de la conception de l'architecture de l'application. Ceci sera présenté plus tard dans la section 5.

### 4.2.3 Tests réalisés

Une fois les principaux composants définis, le prototype a été testé pour s'assurer que tous les composants fonctionnaient correctement ensemble. En outre, il fallait vérifier si le SIM800L fonctionnait correctement avec le SDK de Thingstream, ou s'il causerait les mêmes problèmes que ceux rencontrés l'année précédente. De ce fait, un protocole de test a été mis en place.

Très brièvement, ce protocole consiste à lancer un stress test sur deux cœurs du CPU du Raspberry Pi et, en même temps, de publier un message toutes les 4 minutes sur un topic MQTT en utilisant le SIM800L et la plateforme Thingstream. Pendant le test, un serveur est aussi connecté à la plateforme de Thingstream (IP-Thing) et son but est de répondre à toute demande envoyée par le Raspberry Pi. Après l'envoi du message, le Raspberry Pi attend pendant maximum 2 minutes la réponse du serveur. Si aucune réponse n'est reçue, l'occurrence est enregistrée sur un fichier .txt. Le diagramme de séquence 4.8 montre comment fonctionne l'envoi d'un message pendant une expérimentation. Le test a une durée de 4 heures et 45 minutes. Après cette période, des mesures de température supplémentaires sont prises pendant 15 minutes. Ce protocole a été automatisé à l'aide de scripts python afin de garantir sa reproductibilité. Dans l'annexe B se trouvent plus de détails concernant le matériel nécessaire, le protocole et la démarche à suivre.

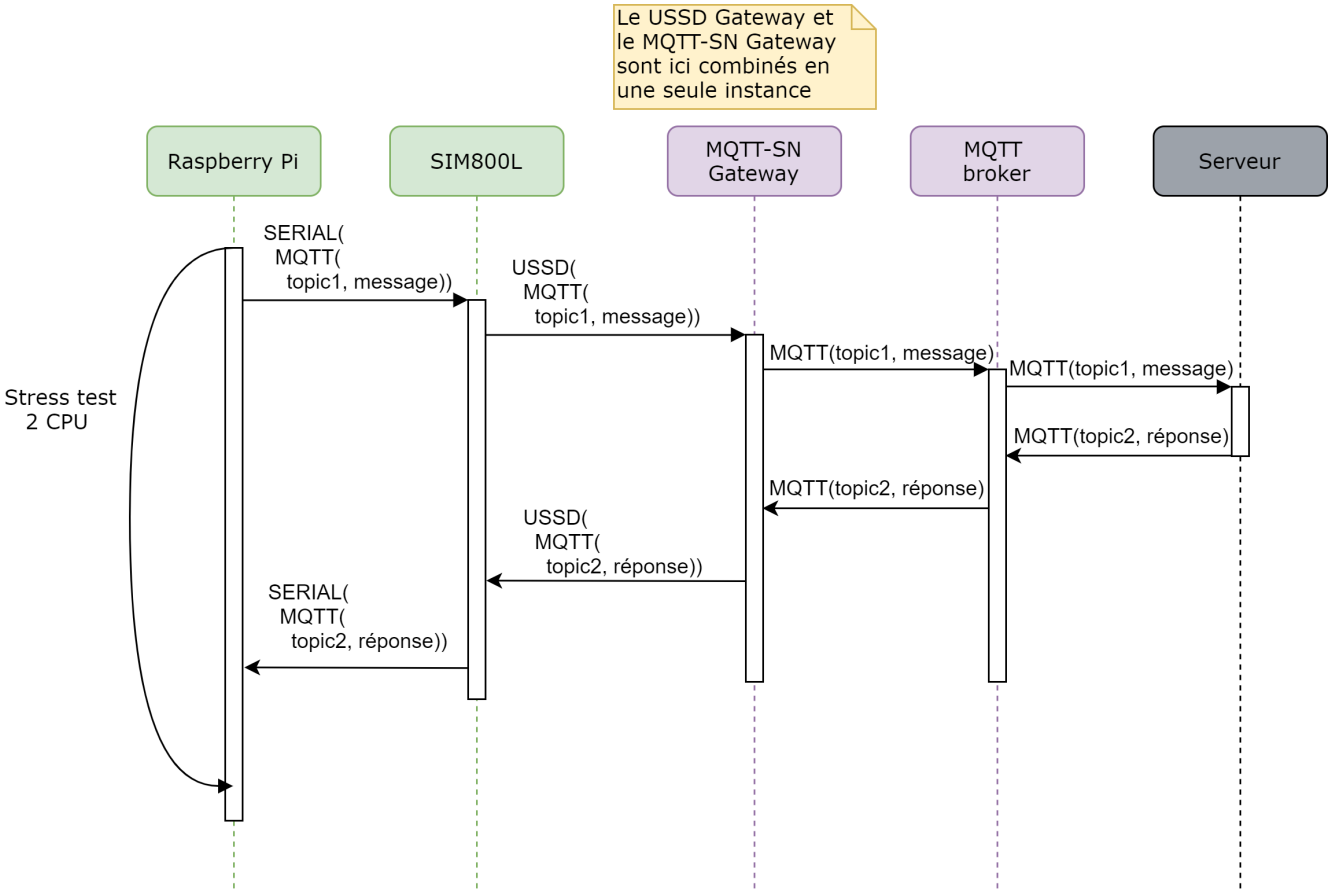


Figure 4.8: Diagramme de séquence de la publication d'un message pendant un test

Pendant le test, plusieurs métriques sont surveillées afin d'évaluer le fonctionnement du prototype. En premier lieu, il y a la température du CPU qui est mesurée toutes les 30 secondes. Ensuite, la température de deux zones distinctes est mesurée à l'aide de capteurs externes de température. Ces deux zones sont indiquées sur la figure 4.8 et correspondent aux deux zones où la température est la plus élevée à l'intérieur du boîtier. Ces trois mesures de température servent principalement à analyser les capacités de refroidissement du boîtier, mais aussi pour vérifier si la température a un impact sur les performances du SIM800L.

Ensuite, il y a les métriques liées au fonctionnement du SIM800L. Les logs produits par le SDK de Thingstream permettent de ressortir deux informations assez importantes : le nombre de messages impossibles à décoder (Receive:parseMessage Warning) et le nombre d'erreurs lors de l'envoi d'un paquet (Protocol:SendPackets Error). La quantité de messages envoyés et reçus par le Raspberry Pi a aussi été enregistrée pour tous les tests, mais cette métrique ne sera pas discutée ici puisque

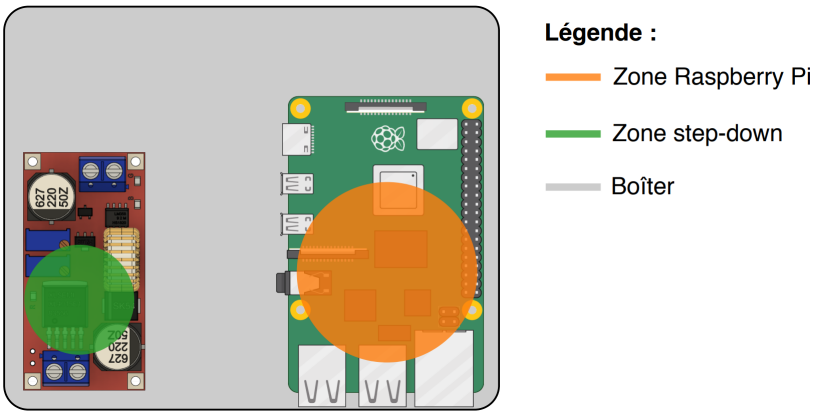


Figure 4.9: Zones où la température a été mesurée lors des tests

cette valeur était la même pour tous les tests. Toutefois, nous pouvons déjà conclure que le SDK de Thingstream est résilient aux erreurs du modem une fois que le nombre d'erreurs ne semble pas influencer les messages envoyés.

Les tests présentés ci-dessous ont été réalisés avec le boîtier du prototype de l'année précédente. Le boîtier se trouvait dans une pièce à température contrôlée entre 19,5°C et 20,5°C. Lors de ces tests, le système d'alimentation avec batterie n'a pas utilisé. Au lieu de cela, un chargeur 12V 1.5A DC a été utilisé pour alimenter le XL4015. Chaque test a été répété trois fois et les résultats correspondent à la moyenne de ces trois essais.

Ce protocole de test a été suivi pour effectuer quatre tests avec des conditions de fonctionnement différentes. Le premier test a été réalisé sans aucun refroidissement actif et avec le step-down à l'intérieur de la boîte. Ce test correspond aux conditions normales de fonctionnement du prototype de l'année dernière.

Ensuite, dans le test 2, le XL4015 a été retiré de la boîte et placé à une distance de 50 cm de celle-ci. Des fils d'une longueur de 60 cm étaient alors utilisés pour relier le XL4015 aux différents composants de la boîte. Ce test avait comme but de vérifier l'impact du XL4015, et en particulier du champ magnétique produit par celui-ci, sur les performances du SIM800L.

Dans le troisième test, le step-down se trouvait à nouveau à l'intérieur du boîtier, mais un ventilateur était utilisé pour souffler de l'air vers l'intérieur de la boîte. Dans ce test, l'objectif était de comprendre à quel point un ventilateur améliore la solution de refroidissement du boîtier. De plus, l'idée était aussi de vérifier si la température a un impact sur les performances du modem.

Finalement, dans le quatrième test, le step-down a été entouré en 10 couches de feuille d'aluminium et placé à l'intérieur de la boîte. Les couches de feuille d'aluminium permettent de créer un blindage magnétique pour protéger le modem.

Le troisième test montre que le ventilateur a un grand impact sur la température à l'intérieur du boîtier. En effet, la température du CPU chute entre 14°C à 15°C après l'ajout du ventilateur. De plus, la température dans le boîtier devient plus homogène. Il est tout à fait prévisible que la température du XL4015 diminue aussi avec l'ajout du ventilateur, ce qui est bénéfique pour la durée de vie du composant. Cependant, le nombre élevé de warnings et d'erreurs pour le troisième test indique que la température n'a pas d'impact sur les performances du SIM800L. (tableau 4.1)

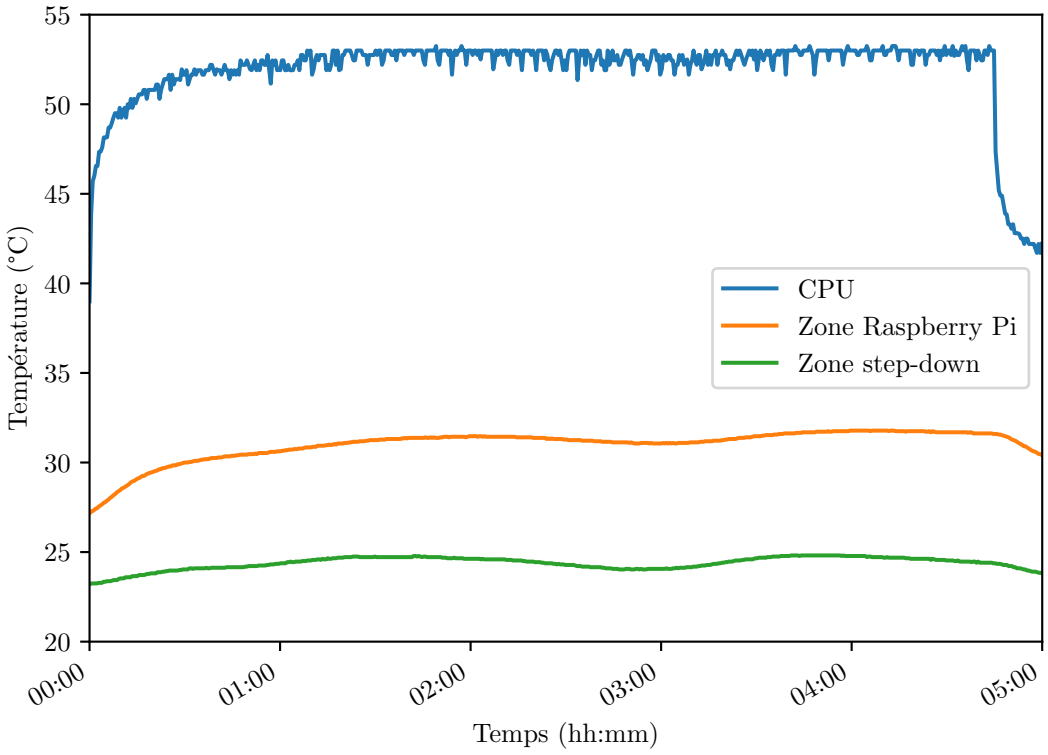


Figure 4.10: **Test 1 :** Évolution de la température dans les tests avec step-down et sans ventilateur

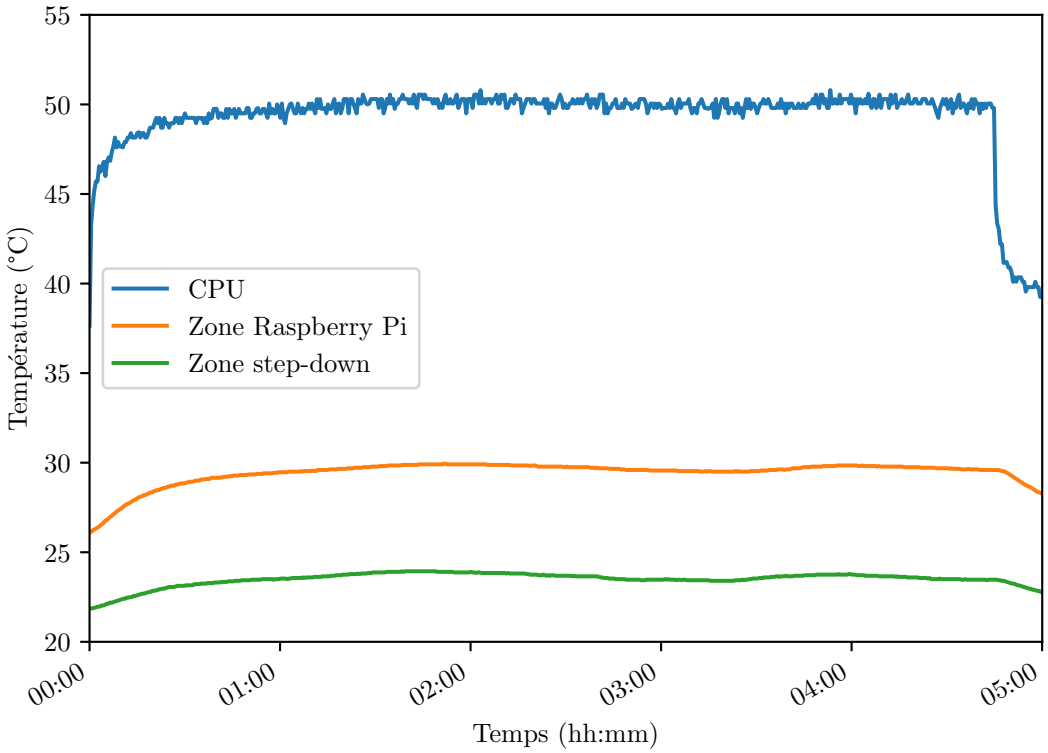


Figure 4.11: **Test 2 :** Évolution de la température dans les tests sans step-down et sans ventilateur

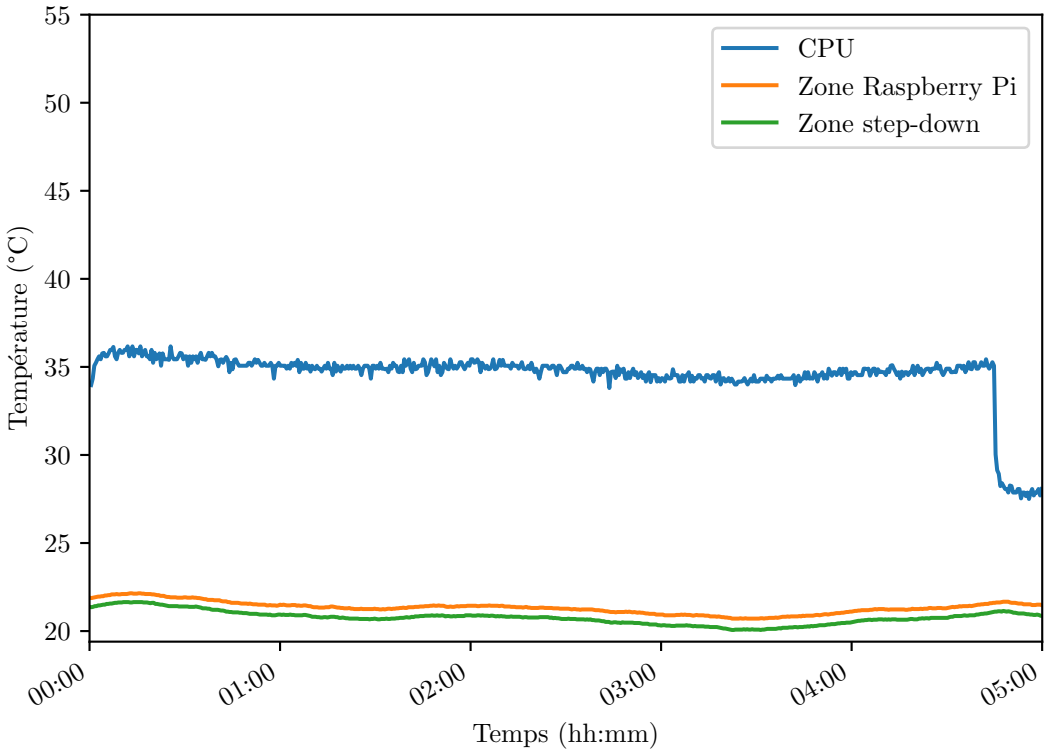


Figure 4.12: **Test 3 :** Évolution de la température dans les tests avec step-down et avec ventilateur

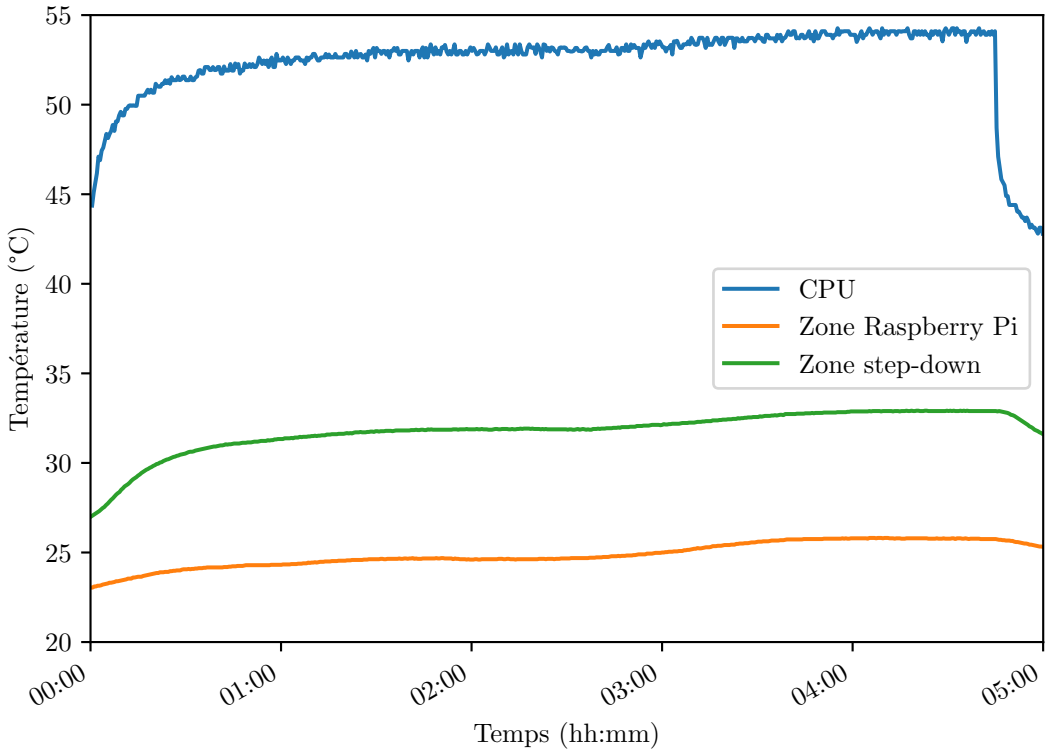


Figure 4.13: **Test 4 :** Évolution de la température dans les tests avec step-down et blindage magnétique mais sans ventilateur

Les résultats des tests 2 et 4 permettent de conclure que le champ magnétique produit par le

| | Test 1 | Test 2 | Test 3 | Test 4 |
|---|---|---|---|---|
| **Protocol:SendPackets Error** | min : 6<br>max : 9<br>moy : 7,33 | min : 2<br>max : 3<br>moy : 2,33 | min : 8<br>max : 13<br>moy : 10,33 | min : 2<br>max : 6<br>moy : 4 |
| **Receive:parseMessage Warning** | min : 4<br>max : 5<br>moy : 4,33 | min : 0<br>max : 0<br>moy : 0 | min : 4<br>max : 9<br>moy : 6,33 | min : 0<br>max : 0<br>moy : 0 |

Table 4.1: Statistiques loggés par le SDK de Thingstream pour chaque test

step-down est la cause des défaillances au niveau du SIM800L. En effet, lors de ces deux tests, le Raspberry Pi et le modem n'ont plus eu de problème à échanger des messages. De plus, le nombre d'erreurs pendant de la transmission de paquets diminue aussi considérablement.

Les résultats des tests ci-dessous permettent de tirer les conclusions suivantes :

- Le prochain boîtier devrait avoir un ventilateur intégré afin de mieux gérer la température des composants.
- Dans la mesure du possible, le step-down doit être éloigné du SIM800L. Un blindage magnétique peut être utilisé en supplément lorsque la distance entre les deux composants n'est pas suffisante.

Les deux points présentés ci-dessous définissent la base du nouveau boîtier qui a été spécifiquement créé pour loger les différents composants de ce projet. Le boîter sera présenté dans la section suivante.

### 4.2.4 Résultat

À la suite des choix présentés ci-dessus, les principaux composants du prototype sont :

- Raspberry Pi 2B
- SIM800L
- XL4015
- Circuit imprimé réalisé l'année précédente

Comme expliqué précédemment, une boîte a été conçue pour abriter les composants. Pour ce faire, le boîtier a d'abord été modélisé en 3D sur Fusion 360. Ensuite, il a été produit en ayant recours à l'impression en 3D. Ce procédé de fabrication a permis de concevoir le boîtier en plus ou moins une demi-journée.

Dans cette nouvelle boîte, le step-down a été éloigné le plus possible du SIM800L et une paroi entre les deux composants a aussi été ajoutée. Un matériau d'isolation magnétique peut être additionné sur la paroi afin de limiter l'impact du champ magnétique sur le modem. Un ventilateur de 10mm de diamètre a également été ajouté afin d'avoir une meilleure solution de refroidissement. Veuillez noter que le ventilateur est placé de façon à aspirer l'air vers l'intérieur. Le flux d'air quittant le ventilateur est turbulent et cela permet qu'il se propage dans un peu près toutes les zones à

Figure 4.14: Vue des faces avant et supérieure avec le couvercle

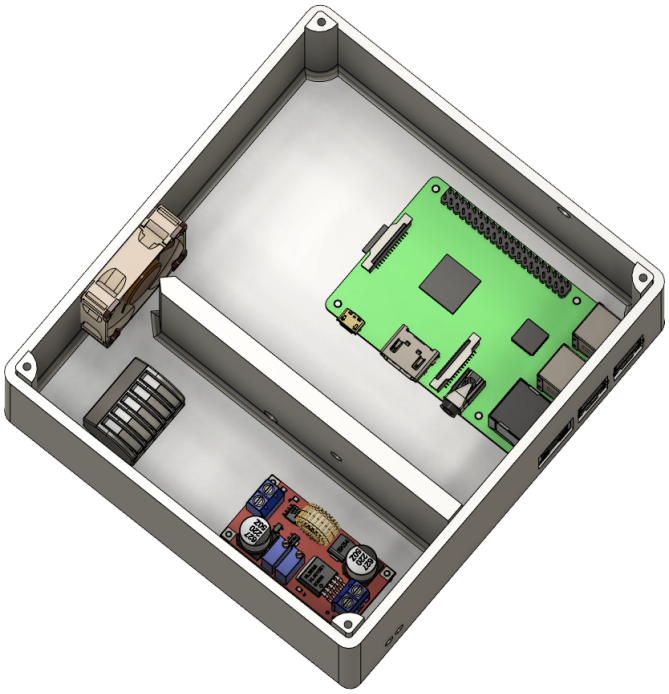

Figure 4.15: Vue de la face supérieure sans le couvercle

l'intérieur de la boîte [39]. Le résultat de la modélisation 3D est visible sur les figures 4.14, 4.15 et 4.16.

Afin de valider le boîtier, le protocole de test présenté dans la section 4.2.3 a été à nouveau employé. Dans le premier test (test 5), aucun matériau de blindage magnétique n'a été utilisé. Toutefois, la distance accrue entre le step-down et le SIM800L suffit déjà pour que la communication entre le Raspberry et le modem ait lieu sans aucun problème. Ajouter des feuilles de papier d'aluminium sur la paroi permet de diminuer encore plus l'impact du champ magnétique puisque le nombre d'erreurs est divisé par deux. (voir test 6 sur 4.2). En particulier, cette dernière solution s'est montrée presque aussi efficace qu'éloigner le step-down 50 cm du boîtier.

La solution de refroidissement de ce boîtier est également très performante puisque la température

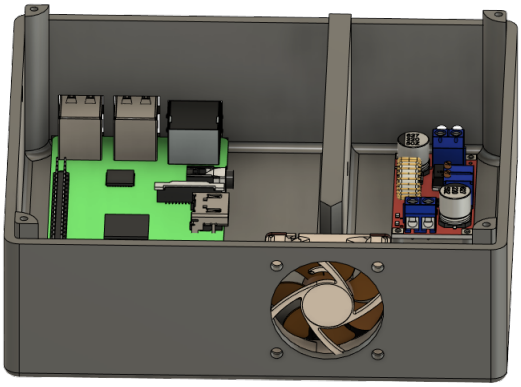

Figure 4.16: Vue de la face arrière sans le couvercle

dans les deux zones (Raspberry Pi et step-down) est tout à fait identique. Ceci montre donc que l'importance du boîtier ne peut pas être négligée. En effet, avoir un boîtier adapté aux besoins des composants est essentiel afin d'assurer le bon fonctionnement de ces derniers et de prolonger leur durée de vie.

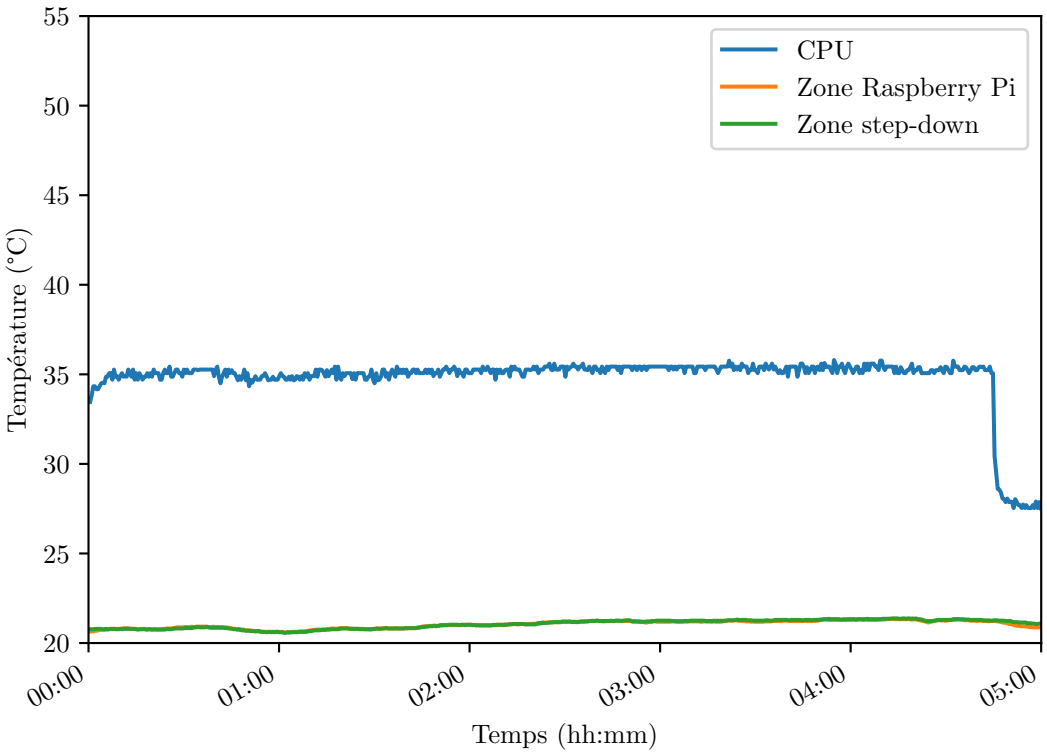


Figure 4.17: **Test 5 :** Évolution de la température dans les tests réalisés dans le nouveau boîtier

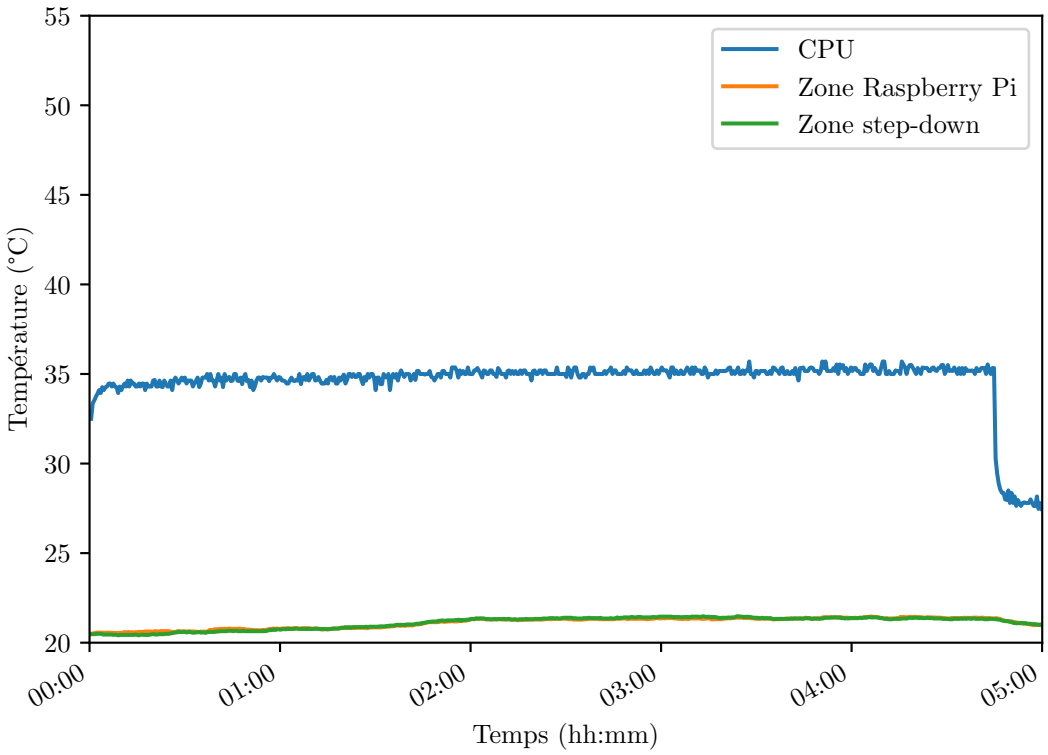


Figure 4.18: **Test 6 :** Évolution de la température dans les tests réalisés dans le nouveau boîtier avec blindage magnétique

| | **Test 5** | **Test 6** |
|---|---|---|
| **Protocol:SendPackets Error** | min : 4<br>max : 5<br>moy : 4,33 | min : 2<br>max : 3<br>moy : 2,67 |
| **Receive:parseMessage Warning** | min : 0<br>max : 0<br>moy : 0 | min : 0<br>max : 0<br>moy : 0 |

Table 4.2: C

# Chapter 5

# L'application

## 5.1 Client

### 5.1.1 Architecture

### 5.1.2 Base de données

### 5.1.3 Implémentation des fonctionnalités

#### Mappage du réseau local

#### Surveiller la présence des divers composants

#### Monitoring du serveur

#### Envoie des données

### 5.1.4 Sécurité

## 5.2 Serveur

### 5.2.1 Architecture

### 5.2.2 Base de données

### 5.2.3 Implémentation des fonctionnalités

### 5.2.4 Sécurité

# Chapter 6

# Tests et résultats

# Chapter 7

# Conclusion

# Bibliography


[1] Knud Lasse Lueth. State of the iot 2018: Number of iot devices now at 7b – market accelerating. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/, Aug 2018. [Accédé le 28-Avril-2020].

[2] Frederic Vannieuwenborg, Sofie Verbrugge, and Didier Colle. Choosing iot-connectivity? a guiding methodology based on functional characteristics and economic considerations. *Transactions on Emerging Telecommunications Technologies*, 29, 10 2017.

[3] Ramon Sanchez-Iborra and Maria-Dolores Cano. State of the art in lp-wan solutions for industrial iot services. *Sensors*, 16(5):708, 2016.

[4] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. Overview of cellular lpwan technologies for iot deployment: Sigfox, lorawan, and nb-iot. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 197–202. IEEE, 2018.

[5] Brandon Foubert and Nathalie Mitton. Long-range wireless radio technologies: A survey. *Future Internet*, 12:13, 01 2020.

[6] David Feugey. Comment la 5g part à l'assaut de l'internet des objets. https://www.orange-business.com/fr/blogs/usages-dentreprise/mobilite/comment-la-5g-part-l-assaut-de-l-internet-des-objets, Dec 2016. [Accédé le 28-Avril-2020].

[7] Renouvellement du réseau mobile. https://www.swisscom.ch/fr/about/entreprise/portrait/reseau/remplacement-2g.html. [Accédé le 28-Avril-2020].

[8] Cisco. Cisco annual internet report (2018–2023). Technical report, Cisco, 03 2020.

[9] B. Brazil. *Prometheus - Up and Running: Infrastructure and Application Performance Monitoring*. O'Reilly Media, 2018.

[10] Jeffrey D. Case, Mark Fedor, Martin Lee Schoffstall, and Chuck Davin. Simple network management protocol (snmp). RFC 1098, RFC Editor, April 1989. http://www.rfc-editor.org/rfc/rfc1098.txt.

[11] Jeffrey D. Case, Mark Fedor, Martin Lee Schoffstall, and James R. Davin. Simple network management protocol (snmp). STD 15, RFC Editor, May 1990. http://www.rfc-editor.org/rfc/rfc1157.txt.

[12] Tech hub guides - monitoring at scale - monitoring architecture. https://developer.lightbend.com/guides/monitoring-at-scale/monitoring-architecture/architecture.html. [Accédé le 29-Avril-2020].

[13] Giedrius Statkevičius. Push vs. pull in monitoring systems. https://giedrius.blog/2019/05/11/push-vs-pull-in-monitoring-systems/, May 2019. [Accédé le 29-Avril-2020].

[14] schkn. Prometheus monitoring : The definitive guide in 2019. https://devconnected.com/the-definitive-guide-to-prometheus-in-2019/, May 2019. [Accédé le 29-Avril-2020].

[15] Faq: Why do you pull rather than push? https://prometheus.io/docs/introduction/faq/#why-do-you-pull-rather-than-push? [Accédé le 29-Avril-2020].

[16] Kiran Oliver. Prometheus and the debate over 'push' versus 'pull' monitoring. https://thenewstack.io/exploring-prometheus-use-cases-brian-brazil/, 2019. [Accédé le 29-Avril-2020].

[17] Ethan Galsta. Ndoutils documentation version 2.x. https://assets.nagios.com/downloads/nagioscore/docs/ndoutils/NDOUtils.pdf, Mar 2017. [Accédé le 29-Avril-2020].

[18] Zabbix documentation 4.4: Database creation. https://www.zabbix.com/documentation/current/manual/appendix/install/db_scripts. [Accédé le 29-Avril-2020].

[19] Bastien L. Time series database : qu'est-ce que c'est, à quoi ça sert ? https://www.lebigdata.fr/time-series-database-definition, May 2018. [Accédé le 29-Avril-2020].

[20] Zhaofeng Zhou. Key concepts and features of time series databases. https://www.alibabacloud.com/blog/key-concepts-and-features-of-time-series-databases_594734, Apr 2019. [Accédé le 29-Avril-2020].

[21] Sheng Di, Hai Jin, Shengli Li, Jing Tie, and Ling Chen. Efficient time series data classification and compression in distributed monitoring. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 389–400. Springer, 2007.

[22] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *ACM Queue*, 8:20, 01 2010.

[23] Nikos Bikakis. Big data visualization tools. *arXiv preprint arXiv:1801.08336*, 2018.

[24] Monthly weather forecast and climate kinshasa, democratic republic of congo. https://www.weather-atlas.com/en/democratic-republic-of-congo/kinshasa-climate. [Accédé le 2-Mai-2020].

[25] Raspberry pi 2 model b. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Accédé le 7-Mai-2020].

[26] Carte de couverture 3g / 4g / 5g airtel, la république démocratique du congo. https://www.nperf.com/fr/map/CD/-/10496.Airtel/signal/?ll=-2.6358509667379706&lg=27.366943359375004&zoom=6. [Accédé le 12-Mai-2020].

[27] Carte de couverture 3g / 4g / 5g orange mobile, la république démocratique du congo. https://www.nperf.com/fr/map/CD/-/5932.Orange-Mobile/signal/?ll=-3.9957805129630253&lg=29.047851562500004&zoom=5. [Accédé le 12-Mai-2020].

[28] K Krithiga Lakshmi, Himanshu Gupta, and Jayanthi Ranjan. Ussd—architecture analysis, security threats, issues and enhancements. In *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions)(ICTUS)*, pages 798–802. IEEE, 2017.

[29] Marcin Hajdul and Arkadiusz Kawa. Global logistics tracking and tracing in fleet management. pages 191–199, 03 2015.

[30] M. Becker, K. Li, K. Kuladinithi, and T. Pötsch. Transport of coap over sms, ussd and gprs. https://tools.ietf.org/html/draft-becker-core-coap-sms-gprs-03, Feb 2013. [Accédé le 14-Mai-2020].

[31] Trevor Perrier, Brian DeRenzi, and Richard Anderson. Ussd: The third universal app. In *Proceedings of the 2015 Annual Symposium on Computing for Development*, pages 13–21, 2015.

[32] Africa's talking : Ussd. https://africastalking.com/ussd. [Accédé le 14-Mai-2020].

[33] Cell mapper : Airtel ug towers location. https://www.cellmapper.net/map?MCC=630&MNC=2&type=GSM&latitude=-3.339888039022128&longitude=30.711752565964886&zoom=4.9229320495685505&showTowers=true&showTowerLabels=true&clusterEnabled=true&tilesEnabled=true&showOrphans=false&showNoFrequencyOnly=false&showFrequencyOnly=false&showBandwidthOnly=false&DateFilterType=None&showHex=false&showVerifiedOnly=false&showUnverifiedOnly=false&showLTECAOnly=false&showENDCOnly=false&showBand=0&showSectorColours=true. [Accédé le 14-Mai-2020].

[34] 5a 180khz 36v buck dc to dc converter xl4015 : Datasheet. http://www.xlsemi.com/datasheet/XL4015%20datasheet.pdf. [Accédé le 14-Mai-2020].

[35] Mqtt: Frequently asked questions. http://mqtt.org/faq. [Accédé le 14-Mai-2020].

[36] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, 1:2, 2013.

[37] Thingstream in¿flow 2019: Intro & thingstream in a nutshell. https://info.thingstream.io/hubfs/inflow%202019%20presentations/Thingstream%20in_flow%202019%20round%20up.pdf/. [Accédé le 11-Mai-2020].

[38] Data flow manager: Flow-based visual programming. https://thingstream.io/products/data-flow-manager/. [Accédé le 12-Mai-2020].

[39] Providing sufficient cooling - electronic cooling and sensible heat. https://www.rs-online.com/designspark/why-do-we-need-airflow-electronic-cooling-sensible-heat, Nov. 2017. [Accédé le 14-Mai-2020].

# Appendix A

# Tableau comparatif des réseaux

| | **LoRaWAN** | **Réseaux mobiles (2G/3G/4G/5G)** | **Sigfox** | **Satellite** | **Wi-Fi** | **Zigbee** | **Bluetooth** | **Réseaux mobiles IoT (NB-IoT, LTE-M)** |
|---|---|---|---|---|---|---|---|---|
| **Débit** | 0.3 - 50kbps | EDGE: 384 Kbps<br>HSPA+: 42 Mbps<br>LTE-A: 1 Gbps<br>5G: 10 Gbps | 100 ou 600 bps | Quelques kbps jusqu'à plusieurs Gbps. Cela dépend de la fréquence de transmission. | Wi-Fi 6: 9.6 Gbps | 250 Kbps | 5.0 : 5 Mbps | NB-IoT: 200 Kbps<br>LTE-M: 1 Mbps |
| **Bande de fréquences** | 868 - 915 MHz | 400 - 3000 Mhz +<br>25 - 39 Ghz (5G) | 862 - 928 Mhz | 1 - 40 GHz | 2.4 GHz / 5 GHz | 868/915/<br>2400 MHz | 2400 Mhz | 800 - 2200 MHz |
| **Canal de communication** | Half-duplex | Full-duplex | Half-duplex | Full-duplex | Half-duplex | Full-duplex | Full-duplex | Half-duplex |
| **Consommation énergétique** | Très Basse | Haute | Très Basse | Haute | Moyenne | Très Basse | Très Basse | Basse |
| **Portée** | 5 - 20 km | 2 km - 35 km | 10 - 40 km | Mondiale | 15 - 300 m | 30 - 100 m | 3 - 30 m | NB-IoT : 1 - 15 km<br>LTE-M : 1 - 11 km |
| **Couverture** | | Mondiale | Europe de l'ouest principalement | Mondiale | Réseau privé | Réseau privé | Réseau privé | |
| **Sécurisé** | Oui (AES 128) | Exploits possibles dans le réseau GSM.<br><br>Nouvelles versions sont plus sécurisées. (Possible aussi d'ajouter dans la couche application) | Non (Possible dans la couche application) | Dépend du standard utilisé | Oui, mais des exploits existent | Oui (AES 128) | Oui (propriétaire) | Oui (propriétaire) |

# Appendix B

# Protocole de test : Température et Communication

Pour rappel, le prototype est constitué des composants suivants :

- Raspberry Pi 2B (plateforme)
- XL4015 (step-down)
- SIM800L (modem)
- Carte SIM Thingstream
- Circuit imprimé permettant de connecter le Raspberry Pi et le SIM800L
- Un boîtier (avec ou sans ventilateur)

En plus du matériel requis pour le prototype, les composants suivants sont également nécessaires afin de réaliser le test :

- Un Raspberry Pi connecté à Internet (le modèle n'a pas d'importance)
- Deux capteurs de température de type DS18B20
- Une résistance de $4,7k\Omega$

Afin de mieux distinguer les deux Raspberry Pi, le Raspberry Pi 2B sera désormais nommé client. Le Raspberry Pi connecté à Internet sera appelé serveur.

Avant de démarrer le test, il faut connecter les capteurs de température au serveur. La figure illustre comment effectuer ce branchement.
Le premier capteur de température doit être placé à environ 1 cm au-dessus du circuit imprimé qui est placé sur le client. Le deuxième capteur de température doit être placé à plus ou moins 1cm au-dessus du XL4015. La figure 4.2 illustre les régions où les capteurs doivent être placés.

Ensuite, les scripts python doivent être copiés sur le serveur et le client si cela n'a pas encore été fait. Pour ce faire, vous pouvez utiliser le client SFTP de votre choix. Les fichiers *automated_test.py*, *client_test.py*, *cpu_temp.py* et *MA2_Communication_server.jar* doivent être copiés sur le client. Le fichier *server_test.py* doit être copié sur le serveur. Sur ce dernier fichier, veuillez configurer les variables *CLIENT_ID*, *USERNAME* et *PASSWORD* avec des paramètres permettant de se connecter à la plateforme de Thingstream. (Ces valeurs se trouvent sur votre portail de Thingstream)

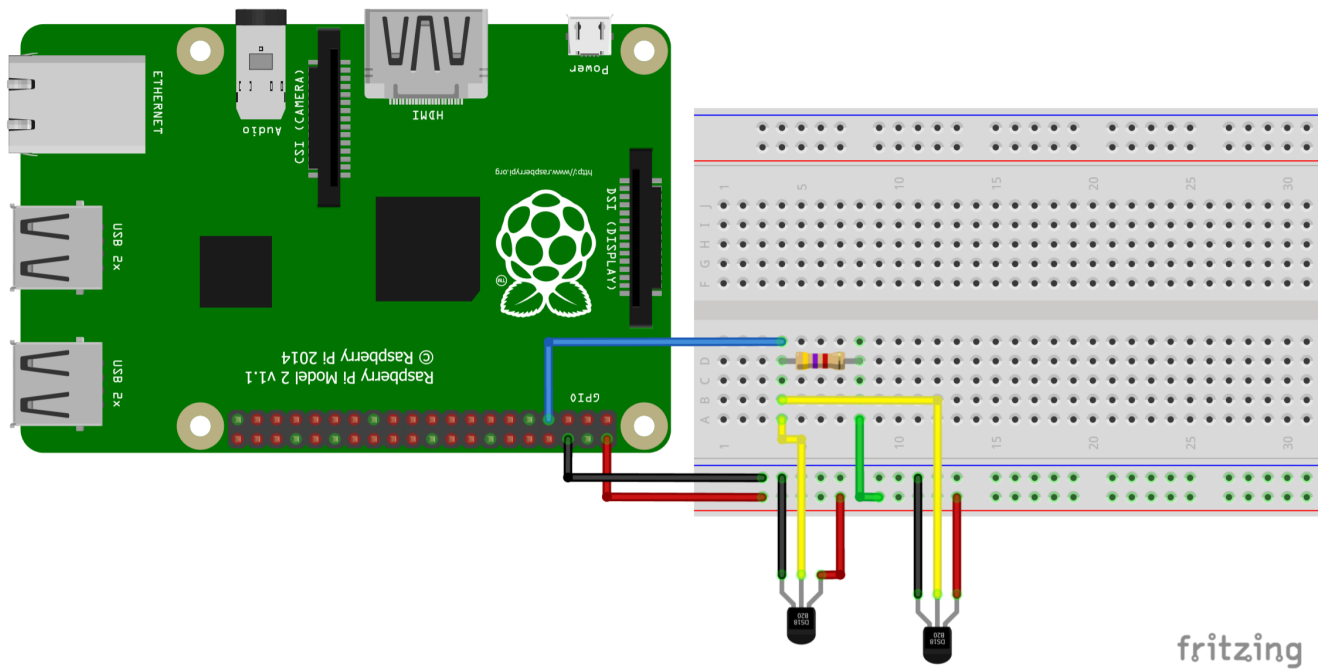


Figure B.1: Schéma des connexions des capteurs de température au Raspberry serveur

Pour lancer le test du côté du client, seulement le programme dans *automated_test.py* doit être exécuté. Ce programme se chargera de lancer tous les autres programmes requis pour le test. Cependant, pour lancer le programme, les paramètres suivants doivent être passés en argument lors de l'exécution :

- -r [*int*] : durée totale du test en seconds
- -t [*int*] : temps en seconds entre chaque mesure de la température du CPU
- -m [*int*] : temps en seconds entre la publication de chaque message
- -f [*str*] : nom du fichier avec les résultats

De façon similaire, plusieurs paramètres sont requis pour exécuter le *serser_test.py*. Les arguments sont les suivants :

- -r [*int*] : durée totale du test en seconds
- -t [*int*] : temps en seconds entre chaque mesure prise pas les capteurs DS18B20
- -f [*str*] : nom du fichier avec les résultats

Le protocole de test peut être divisé en deux parties. La première partie a comme durée la valeur passée avec le paramètre r moins 900 seconds. Les 15 dernières minutes (900 seconds) correspondent à la deuxième partie du test. Cette première partie est composée par la succession d'étapes suivantes.

1. Deux cœurs du CPU du client sont soumis à un stress test. Ce stress test se prolonge pendant toute la première partie du protocole de test.

2. En parallèle, le prototype publie un message sur le topic *device/{identity}/post* toutes les -m secondes. Après avoir publié son message, le client attend pendant 2 minutes maximum une réponse du serveur.

3. Le serveur reçoit le message publié par le client et vérifie si ce message est valide. Le résultat de cette vérification est enregistré sur le fichier avec le nom du paramètre -f. Ensuite, le serveur publie une réponse sur le topic *device/{identity}/startup*.

4. Si le client reçoit la réponse du serveur endéans les 2 minutes à la suite de l'envoi du message, alors la réponse du serveur est écrite sur le fichier de nom -f. Dans le cas contraire, un message d'erreur est écrit au lieu du message.

5. Pendant que les étapes précédentes ont lieu, le client enregistre la température du CPU toutes les -t secondes sur un deuxième fichier de nom "temps_ + -f".

6. Pareillement, le serveur enregistre aussi les températures mesurées avec les capteurs de température sur un fichier de nom "temps_ + -f". Ces mesures sont aussi effectuées toutes les -t secondes.

Lors des 15 dernières minutes, seulement des mesures de température sont effectuées. Il est ainsi plus facile de comparer la façon dont le boîtier ou les différentes solutions de refroidissement sont capables de libérer la chaleur des différents composants. Lors de ce mémoire, tous les tests effectués en suivant ce protocole ont été réalisés à l'aide des commandes suivantes :

Listing B.1: Commande client

```
python3 automated_test.py -r 18000 -t 30 -m 240 -f client_test.txt
```

Listing B.2: Commande serveur

```
python3 server_test.py -r 18000 -t 30 -f server_test.txt
```

Ceci veut donc dire que les tests avaient une durée de 5 heures, les mesures de température étaient effectuées une fois toutes les 30 secondes et un message était publié par le client une fois toutes les 4 minutes. Les deux commandes doivent être lancées l'une après l'autre sans aucun ordre particulier.