



Système intégré de monitoring pour centre hospitalier en Afrique

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

Remerciements

En tout premier lieu, je voudrais remercier le directeur de ce mémoire, le Professeur François Quittin, et mon co-promoteur, le Professeur Antoine Nonclercq, pour leur disponibilité, leurs conseils et les connaissances techniques qu'ils ont apportées à ce mémoire.

Je tiens également à remercier mon superviseur, Quentin Delhaye, de m'avoir tant aidé tout au long de ce mémoire et d'avoir suivi de près tous les progrès réalisés.

Je voudrais aussi remercier Loïc Vaes, chef de projet chez AEDES, et Erica Berghman, ULB-Coopération, de m'avoir accordé l'opportunité de travailler sur ce mémoire.

Je tiens à témoigner toute ma gratitude envers mon ami, Jacky Trinh, et mon frère, Tiago Correia, pour avoir pris le temps de relire ce mémoire et pour tous les conseils qu'ils m'ont donnés concernant son style.

Enfin, un grand merci à mes parents qui m'ont toujours encouragé tout au long de mes études, surtout dans les moments plus difficiles.

Résumé

L'Organisation mondiale de la Santé a déclaré que les systèmes d'information hospitaliers sont un des composants principaux d'une structure de santé. CERHIS est un système d'information qui a été développé pour répondre aux besoins particuliers des centres hospitaliers en Afrique. L'infrastructure de CERHIS est constituée de plusieurs composants, dont des tablettes Android, un serveur local et un système d'alimentation sans interruption. Afin d'assurer l'opérabilité de CERHIS, les différents éléments doivent être surveillés en permanence et les techniciens doivent être alertés en cas d'une défaillance. C'est ce dernier problème que ce mémoire essaye de tacler en créant une solution de monitoring basée sur un Raspberry Pi. Cependant, les conditions particulières de l'environnement imposent certaines contraintes sur cette même solution. En effet, le dispositif doit pouvoir contacter des techniciens par SMS et envoyer des informations sur un serveur distant, mais sans avoir accès à une connexion Internet filaire. En outre, le système doit aussi posséder une batterie pour que le monitoring continue à se faire même en cas de coupure du réseau électrique. La solution présentée ici combine un SIM800L avec le Raspberry Pi de telle sorte que ce dernier puisse se connecter aux réseaux mobiles pour transmettre des données. Thingstream, une plateforme *IoT Communication-as-a-Service*, a été exploitée afin de gérer la connectivité et l'échange d'informations entre SIM800L et le serveur distant. En ce qui concerne le logiciel de monitoring, une solution de pointe a été développée suivant une architecture de microservices et basée sur l'application de monitoring Prometheus couplé à l'outil de visualisation de données Grafana. Les tests effectués avec le prototype ont été très prometteurs et permettent d'établir la direction à prendre à l'avenir pour l'améliorer.

Mots-clés : CERHIS, Système de Monitoring, Prometheus, Microservices

Table des matières

1	Introduction	6
1.1	Contexte du mémoire	6
1.2	Parties prenantes	7
1.3	Projet précédent	7
1.4	Structure du mémoire	8
2	État de l'art	9
2.1	Technologies de communication à distance	9
2.2	Monitoring de l'état d'un serveur linux	12
2.3	Plateformes de visualisation de données	15
3	Cahier des charges	18
3.1	Caractéristiques matérielles du dispositif	19
3.2	Fonctionnalités à implémenter	19
3.3	Priorité des tâches et méthodologie de travail	19
4	Prototype	21
4.1	Le premier prototype (2018-2019)	21
4.1.1	Problèmes détectés	21
4.2	Nouveau Prototype	23
4.2.1	Solutions considérées	23
4.2.2	Thingstream	29
4.2.3	Tests réalisés	30
4.2.4	Résultat	33
5	L'application	39
5.1	Client	39
5.1.1	Architecture	39
5.1.2	Implémentation des fonctionnalités	42
5.1.3	Sécurité	61
5.2	Serveur	61
5.2.1	Base de données	62
5.2.2	Implémentation des fonctionnalités	63
5.2.3	Sécurité	66
6	Tests et résultats	69
6.1	Définition de l'environnement	70
6.2	Résultats	71
6.2.1	Consommation énergétique	71
6.2.2	Utilisation et température du CPU	73
6.2.3	Détection d'alertes et envoi des messages	74

6.2.4	Rapidité de la détection d'alertes	76
7	Conclusion	77
7.1	Suggestions d'amélioration	78
A	Tableau comparatif des réseaux	79
B	Protocole de test : Température et Communication	82
C	Choix de la plateforme (Rapport du projet 2018-2019)	85

Chapitre 1

Introduction

1.1 Contexte du mémoire

De nos jours, les solutions de stockage d'informations au format papier sont remplacées au fur et à mesure par des solutions électroniques. Ces dernières permettent de stocker une énorme quantité de documents sur des supports physiques de petite taille, mais cela est loin d'être leur seul avantage. En effet, les documents électroniques ouvrent la porte à tant d'autres possibilités. Ils facilitent la collaboration, l'échange et la sauvegarde de données et proposent énormément de formats différents pour présenter une grande variété d'informations. De plus, lorsque les informations informatisées se retrouvent sur une base de données, les langages de requête aident à récupérer rapidement les données cherchées.

D'après l'Organisation mondiale de la santé (OMS), les systèmes d'information hospitaliers sont un des six piliers d'une structure de santé [57]. Avoir accès à des informations fiables et solides est indispensable pour conduire toutes les activités liées aux autres composantes du système de santé [55]. Avec ce but est né CERHIS, "un système d'information adapté aux structures de santé situées dans des environnements à faibles ressources". Cette solution est composée de tablettes Android et d'un serveur local qui centralise le stockage de toutes les données. Grâce à un réseau Wi-Fi local, les tablettes peuvent communiquer avec le serveur afin d'échanger les données médicales. En plus de cela, CERHIS possède un appareil d'alimentation mixte¹ dans le but de faire face à d'éventuelles coupures d'électricité.

Toutefois, comme tout système informatique et électronique, les différents composants peuvent tomber en panne, ce qui peut affecter les performances du système sur le long terme. De ce fait, un dispositif de monitoring de l'infrastructure est requis afin de détecter des problèmes et d'avertir les parties concernées le plus rapidement possible. Lors des années académiques antérieures, plusieurs projets ont eu lieu dans l'école polytechnique de Bruxelles dans la finalité de créer ce dispositif de monitoring, dont un projet développé par moi-même l'année dernière.

Le but de ce mémoire est de réunir toutes les connaissances acquises tout au long des programmes précédents pour construire un nouveau prototype capable de surveiller l'installation de CERHIS. En particulier, le prototype de l'année dernière a présenté plusieurs problèmes qui doivent être résolus en vue d'avoir un prototype plus proche du potentiel produit final.

Il faut cependant noter que ce mémoire de fin d'études(MFE) s'encadre dans un projet de coopération et d'aide au développement. Le but n'est pas de proposer une boîte noire avec un guide

1. réseau électrique combiné avec des batteries alimentées par des panneaux solaires

d'instructions sur son utilisation. Au contraire, ce mémoire vise à partager les connaissances requises pour construire un tel produit, ou à pointer le lecteur dans la bonne direction. Les choix effectués seront décrits en détail, ainsi que le fonctionnement du prototype.

Un dernier aspect très important à considérer est que ce mémoire ne se concentre pas sur un seul domaine de l'informatique. En effet, comme il sera présenté plus tard, le prototype encadre une partie de communication utilisant les réseaux sans fil, mais aussi une partie concernant les procédés de monitoring qui est souvent associé au domaine de DevOps. Finalement, il y a également une petite part liée au développement d'un boîtier qui logera tous les composants électroniques.

1.2 Parties prenantes

Ce mémoire étant axé sur la coopération et l'aide au développement, plusieurs parties prenantes sont étroitement associées à sa mise en œuvre. Elles sont présentées ci-dessous :

- **AEDES** : AEDES, ou Agence Européenne pour le Développement et la Santé, “est une société de consultance spécialisée en santé publique”[2] fondée en 1985. Acteur majeur en santé publique, la mission d’AEDES est “l’amélioration de la qualité et de l'accès aux soins de santé partout dans le monde.”[2] Cela est accompli grâce à la mise à disposition de moyens humains et le partage de connaissances. Les contacts avec AEDES ont été réalisés à travers de Loïc Vaes.
- **ULB-Coopération** : “ULB-Coopération est l'ONG de l'Université libre de Bruxelles.” [29] Elle est active dans plusieurs thématiques, dont la santé et les systèmes de santé. “ULB-Coopération s’engage à contribuer à la construction d’une société dans laquelle il fait bon vivre, une société juste, émancipatrice, solidaire, responsable où toutes les citoyennes et tous les citoyens, sans discrimination de genre, sont traité·e·s avec égalité.”[29] Erica Berghman était le point de contact avec ULB-Coopération.
- **Codepo** : Fondée en 2007, Codepo est la cellule de coopération au développement de l’École polytechnique de Bruxelles. Elle propose aux étudiants de Master de participer dans un projet de coopération au développement. Ses principaux objectifs sont la pédagogie, la recherche scientifique et technique et l'éducation au développement. [8] Le Professeur Antoine Nonclercq a représenté la cellule Codepo.

1.3 Projet précédent

Lors des années académiques 2016-2017 et 2017-2018, une solution de monitoring a été créée par des étudiants à l’École polytechnique de Bruxelles. Cette solution utilisait un smartphone Android comme composant principal pour tourner de logiciel de monitoring. Cependant, comme expliqué dans [43], le système d’exploitation Android peut mettre certaines applications en pause pour économiser la batterie du smartphone. Lorsque le smartphone “n'est pas en charge et reste immobile avec l'écran éteint, il passe en mode Doze” [4]. Ce mode de gestion de la batterie ne permet aux applications de fonctionner que pendant de petites périodes appelées fenêtres de maintenance. Le seul moyen de désactiver complètement ce mode est de maintenir le dispositif en charge en permanence. Cela pose un énorme problème puisque le logiciel de monitoring doit être capable de surveiller l'état des différents éléments de façon continue. Faire cela avec un smartphone aurait provoqué une détérioration rapide de sa batterie car elle serait constamment en charge.

Un prototype conçu lors de l'année académique 2018-2019 a essayé de pallier ce problème en remplaçant le smartphone par un Raspberry Pi. Plus de détails sur ce projet seront fournis dans la section 4.1. Ce mémoire s'appuie sur les connaissances acquises tout au long des projets précédents afin de proposer un nouveau prototype.

1.4 Structure du mémoire

Ce mémoire commence par une présentation de l'état de l'art des trois technologies principales de ce mémoire, c'est-à-dire les technologies de communication à distance, les solutions pour le monitoring d'un serveur Linux et les plateformes de visualisation de données. De plus, dans chaque section, il est aussi expliqué la façon dont chacune de ces solutions s'intègre dans ce mémoire.

Ensuite, le cahier des charges établi en début d'année avec AEDES est détaillé. Ce cahier des charges présente non seulement les fonctionnalités du logiciel de monitoring, mais couvre encore certaines propriétés requises afin que le prototype soit adapté à son environnement de fonctionnement. Cette section détaille également l'ordre dans lequel chaque tâche a été réalisée ainsi que la méthodologie adoptée pour accomplir ces dernières.

L'intégralité du prototype construit lors de mémoire est couverte dans le chapitre 4. Tous les composants et solutions considérés sont exposés ainsi que les raisons derrière chaque choix. De plus, le prototype réalisé l'année précédente est également présenté dans cette section puisqu'il a une grande influence sur le prototype conçu cette année. Finalement, le résultat des premiers tests réalisés ainsi que les conclusions à tirer sont détaillés.

Le chapitre 5 présente le logiciel de monitoring implémenté. En particulier, ce chapitre décrit comment les différentes parties du logiciel de monitoring interagissent afin de surveiller l'infrastructure de CERHIS. L'application créée dans ce mémoire est composée des solutions Open Source qui y sont également détaillées. Une section de ce chapitre est aussi réservée à la présentation du serveur distant responsable du stockage de toutes les données envoyées par le logiciel de monitoring.

Valider le bon fonctionnement de toutes les parties constituant le prototype est un élément essentiel de ce mémoire. En effet, il faut comprendre si le résultat de ce mémoire respecte les attentes du cahier des charges ou si un des choix a guidé le prototype dans la mauvaise direction. Le chapitre 6 présente donc en détail les tests réalisés sur l'entièrre de la solution de monitoring produite lors de ce mémoire. De plus, les conclusions à tirer de ces tests sont également explicitées.

Finalement, le dernier chapitre parcourt brièvement les accomplissements de ce mémoire et les prochaines étapes en vue de transformer le résultat en un produit prêt à être déployé dans un environnement de production.

Chapitre 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.

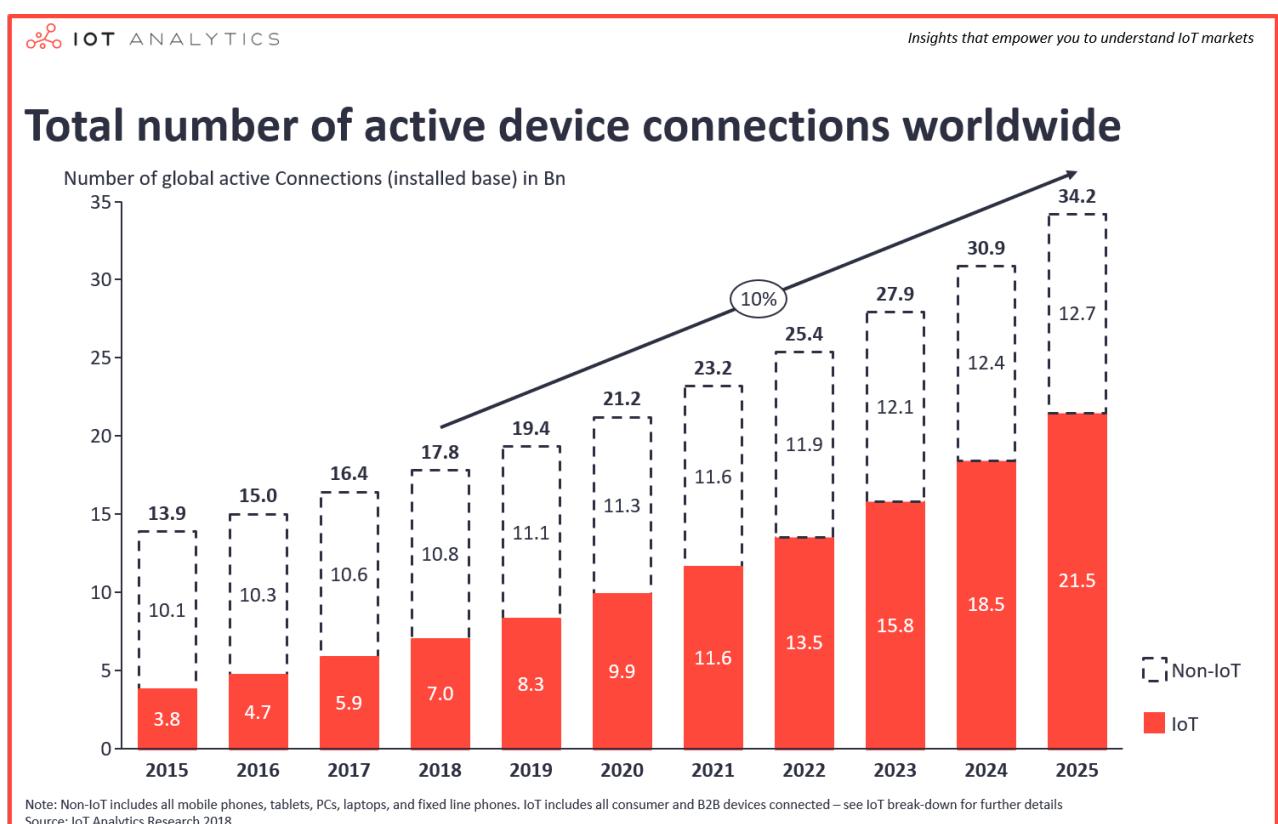


FIGURE 2.1 – Nombre de dispositifs connectés à Internet [53]

Un réseau de communication unique et employable indépendamment du contexte du projet n'existe pas [64]. Toutefois, une multitude de réseaux avec des caractéristiques distinctes sont disponibles, tels que LoRaWAN et GSM. La figure 2.2 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.

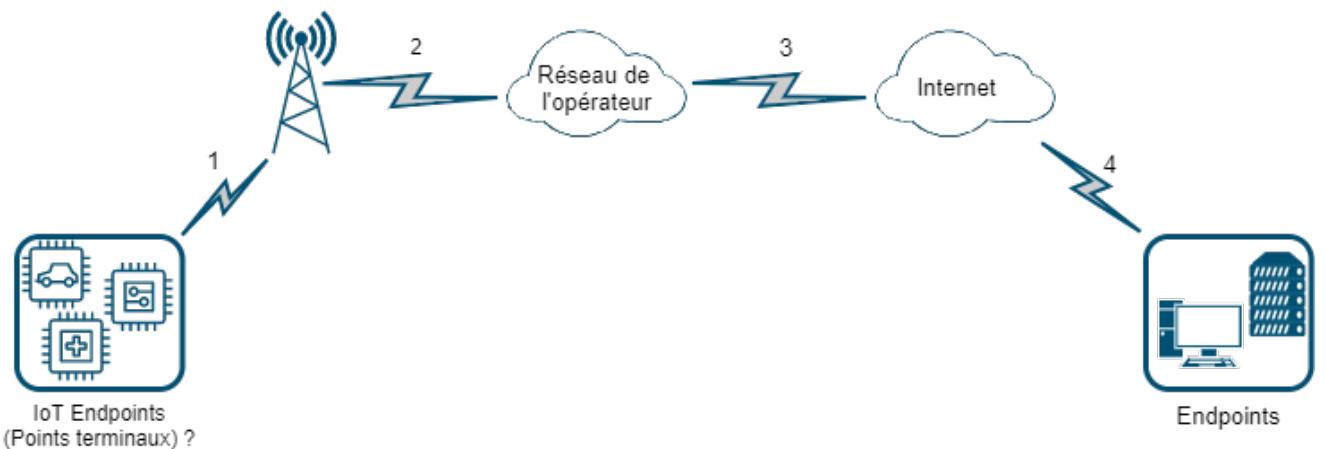


FIGURE 2.2 – Architecture simplifiée d'un réseau avec dispositifs IoT (Basé sur [60, 54])

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. [46] De façon similaire, certaines technologies exploitent la bande ISM¹, 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 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 en fonction de

1. Bande industrielle, scientifique et médicale

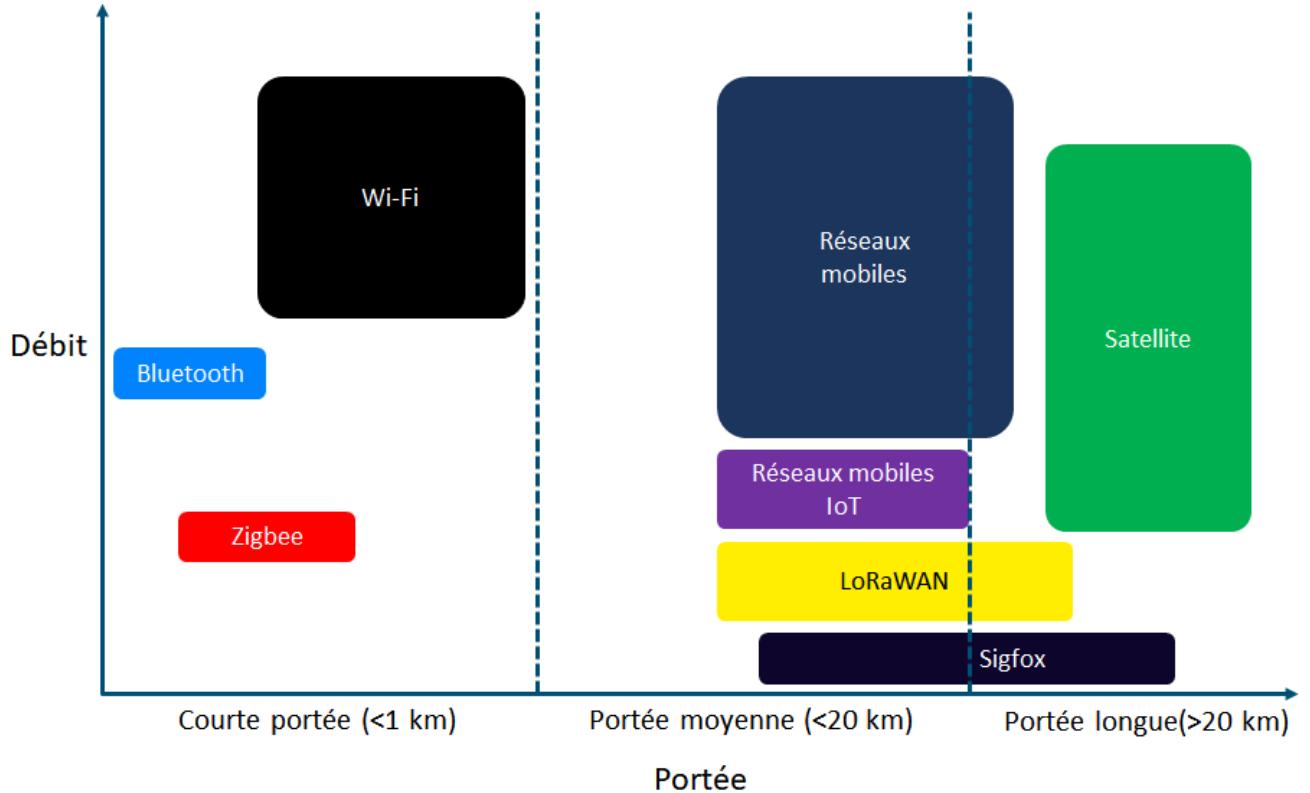


FIGURE 2.3 – Classement des réseaux en fonction de leur portée et débit (Basé sur [54])

certains paramètres tels que la géographie du milieu (urbain ou rural) et des éventuelles interférences. Néanmoins, le débit et la portée sont importants pour classifier ces réseaux et, avec l'aide de la figure 2.3, ces technologies peuvent être classées de la façon suivante [45] :

- **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 [45]. 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 [25]. 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.

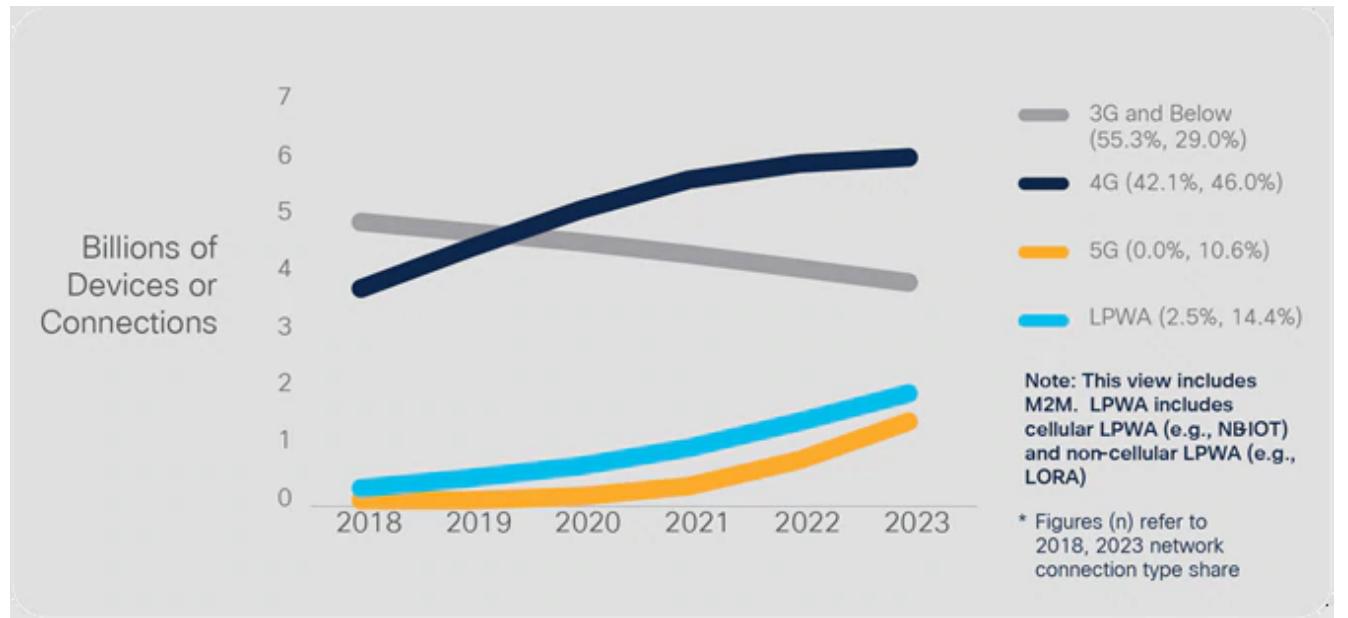


FIGURE 2.4 – Évolution du nombre de dispositifs connectés aux différents réseaux [42]

Comme il sera expliqué plus tard dans ce mémoire, le prototype doit être capable d'utiliser les réseaux de communication à distance. En effet, une connexion filaire à Internet n'est parfois pas disponible dans les structures de santé. Le choix de la technologie plus approprié sera présenté dans la section 4.2.1.

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 événements 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 [37]. 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 [39, 40] permet de monter 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* [37, 27, 63, 27]. 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.

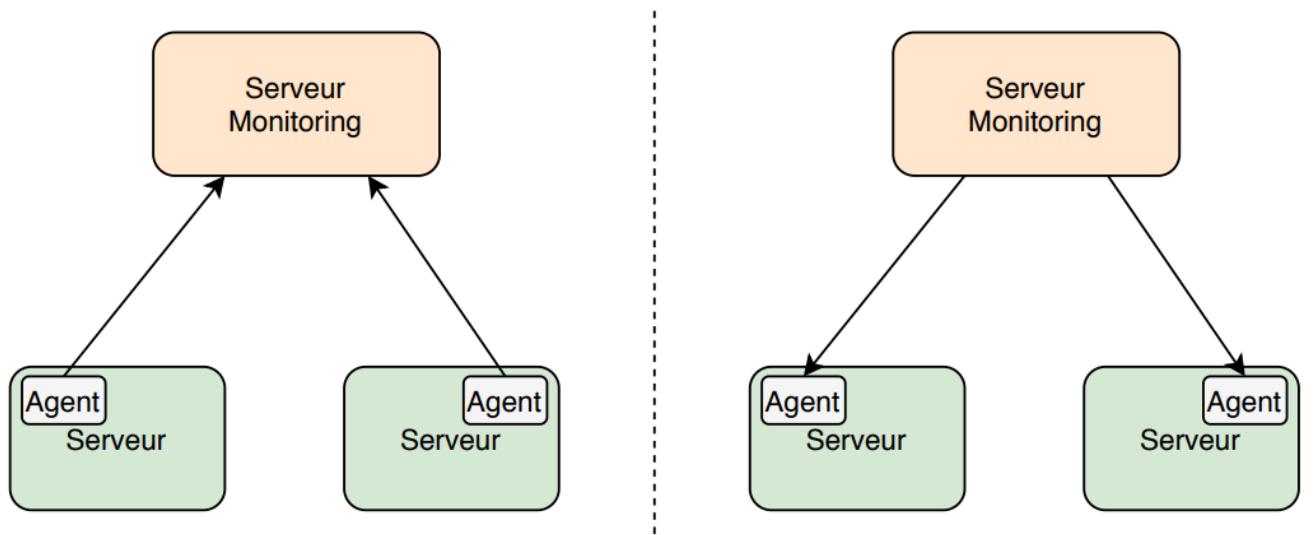


FIGURE 2.5 – **Gauche :** Solution de monitoring avec architecture *push*. **Droite :** Solution de monitoring avec architecture *pull*.

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 batch² de courte durée. [63, 61] 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

2. Un job batch est un programme qui s'exécute sans l'intervention d'un utilisateur de façon régulière. Il permet d'automatiser le traitement d'une grande quantité de données. (par exemple l'automatisation de paiements). L'exécution de plusieurs jobs batch en succession est connue sous le nom de traitement par lots.

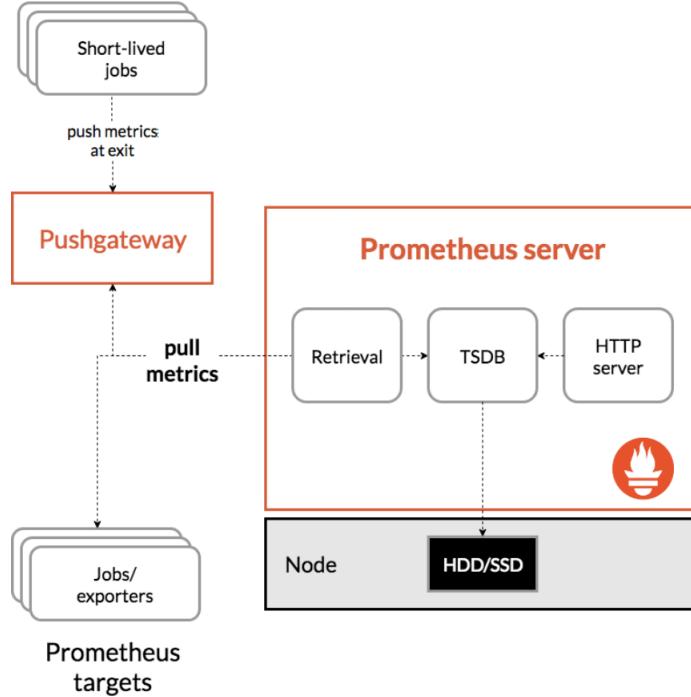


FIGURE 2.6 – Architecture *pull* de Prometheus. Le modèle *push* est supporté grâce au Pushgateway.

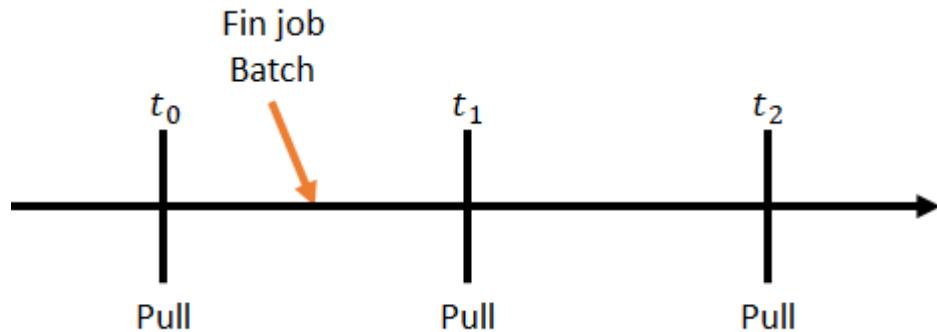


FIGURE 2.7 – Faiblesse du modèle *pull*. Le système de monitoring a raté les informations de la fin du job

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.[10] 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 similaires [56].

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 [47, 30]. Ceci n'est plus le cas pour les solutions plus récentes telles que Prometheus et le TIG Stack³ qui utilisent des bases de données orientées séries temporelles pour la persistance des données.

3. Le TIG stack est la combinaison de Telegraf, InfluxDB et Grafana pour créer un outil de monitoring

Cette sorte de base de données, comme son nom indique, est mieux optimisée pour des valeurs horodatées [51]. En effet, ces bases de données possèdent les propriétés suivantes [65] :

- 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. [44]
- Toutes les requêtes sont effectuées sur base d'un intervalle de 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).

Le but principal de ce mémoire est de construire un système de monitoring pour CERHIS. Il est donc important de comprendre ce qui se fait actuellement dans l'industrie et quels types de solutions sont disponibles à l'heure actuelle. Le logiciel de monitoring implémenté lors de ce mémoire sera décrit en détail dans la section 5.1.1.

2.3 Plateformes de visualisation de données

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. [49]

Les outils de visualisations de données sont alors des solutions software qui aident à accomplir cet objectif. [36] 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 événement, 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 [49] propose différentes méthodes et graphiques pour observer les différents types de données existants.

Comme présenté dans [50], la visualisation de données doit respecter les trois principes suivants :

- **Fiables (Dignes de confiances)** : Ce principe est fondamental, nous devons pouvoir faire confiance aux données qui nous sont présentées. Aucune manipulation doit être effectué sur la façon dont les données sont affichées afin de manipuler la perception de l'observateur. Par exemple, l'axe vertical d'un graphique à barres ne doit jamais être tronqué.
- **Accessibles** : La visualisation des données doit aider les observateurs à mieux comprendre et interpréter les données. Le contexte des observateurs est essentiel pour que la visualisation puisse être adaptée à leurs besoins. Cependant, cela doit être fait avec prudence et sans

avoir un impact sur les données. Par exemple, un sujet complexe ne doit pas être simplifié à l'excès.

- **Élégante** : Une visualisation élégante des données doit captiver et retenir l'attention des observateurs. Toutefois, cela ne doit jamais compromettre les deux autres principes présentés ci-dessus.

Avec l'arrivée de l'ère des Big Data, les technologies de visualisation sont devenues d'autant plus importantes. En effet, la quantité de données à traiter est devenue tellement importante qu'un humain n'est plus capable de le faire. Pour résoudre ce problème, de nouvelles technologies sont émergées, 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 différentes visualisations de données. L'utilisation de ces librairies demande des compétences en programmation.
- Ensuite, il y a les outils purement dédiés à la visualisation de données tels que Grafana (fig. 2.8) ou Google Data Studio. À partir d'une interface web, ces outils permettent de créer différentes visualisations en utilisant seulement des langages de requête comme SQL. Certains de ces outils sont capables d'aider l'utilisateur à formuler des requêtes. (Query Builders)
- Finalement, il y a les outils de Business Intelligence. Généralement, ils proposent les mêmes fonctionnalités qu'un outil de visualisation de données. Cependant, en plus de cela, ils permettent également de traiter les données et d'effectuer des prédictions grâce à celles-ci. Un exemple d'un tel outil est Tableau (fig. 2.9) ou Power BI.

Une bonne visualisation des données est essentielle pour avoir une solution de monitoring. En effet, une bonne visualisation permet de mieux évaluer l'état du système surveillé et d'identifier la source d'un problème plus rapidement. La section 5.1.1 couvrira la manière dont une solution de visualisation de données a été intégrée au logiciel de monitoring.



FIGURE 2.8 – Exemple de visualisation de données avec Grafana

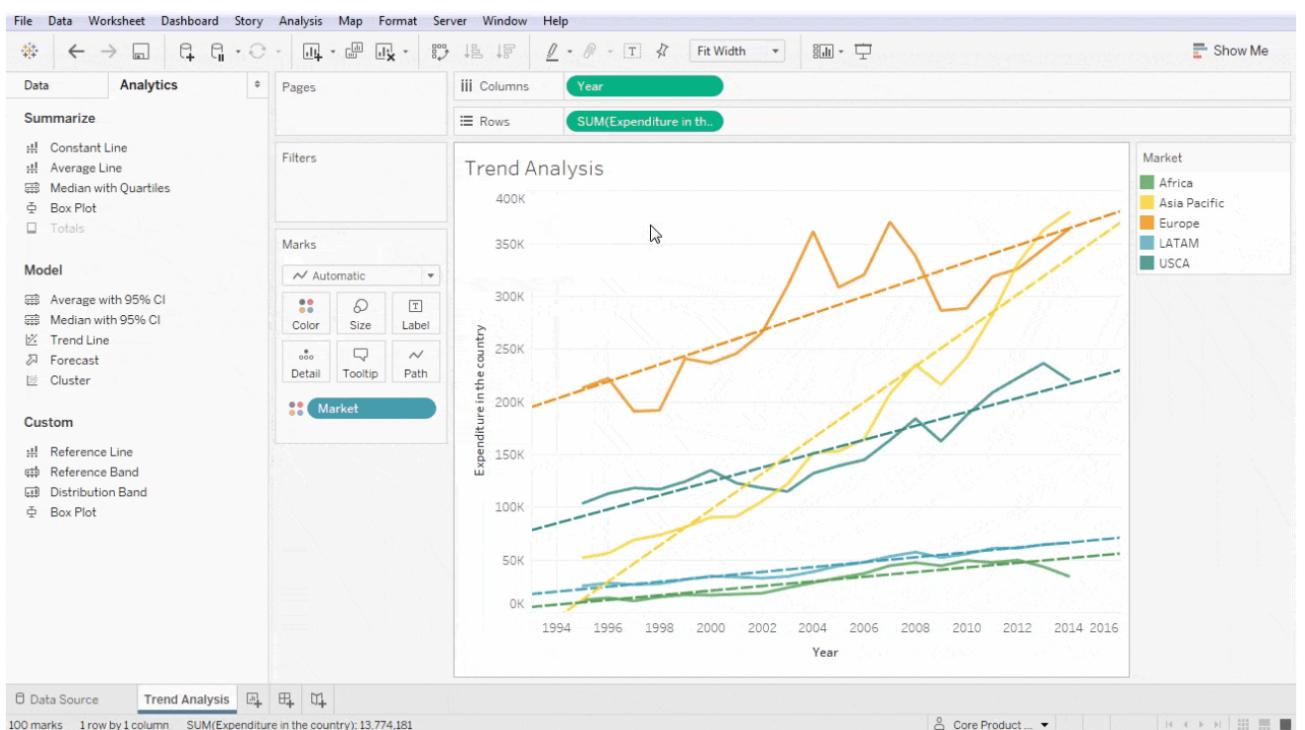


FIGURE 2.9 – Exemple de visualisation et traitement de données avec Tableau [26]

Chapitre 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 portant principalement sur le matériel, y compris 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 logiciel qui permet aux différents composants de s'interfacer. Ensuite, une deuxième partie comprenant juste le logiciel 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).

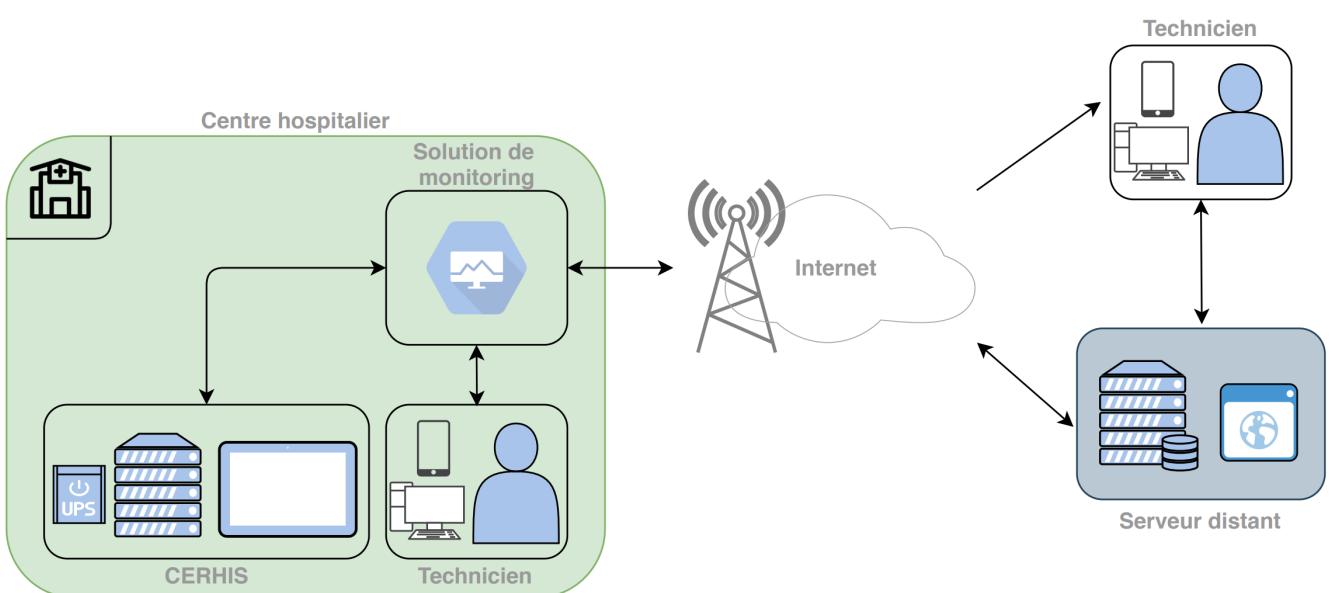


FIGURE 3.1 – Architecture simplifiée de la solution de monitoring

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 élevée. 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. [20]
- 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 (viser 200€ maximum).
- 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 logiciel 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.
 - Envoi 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. 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.

Chapitre 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.

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 interfaçer 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. Le MCP3008 était connecté à la batterie de CERHIS dans le but de monitoner la tension électrique de celle-ci. Dans ce premier essai, le serveur distant 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.

Ce prototype a été testé pendant plusieurs semaines¹ à Goma par un étudiant de l'ISIG². 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

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

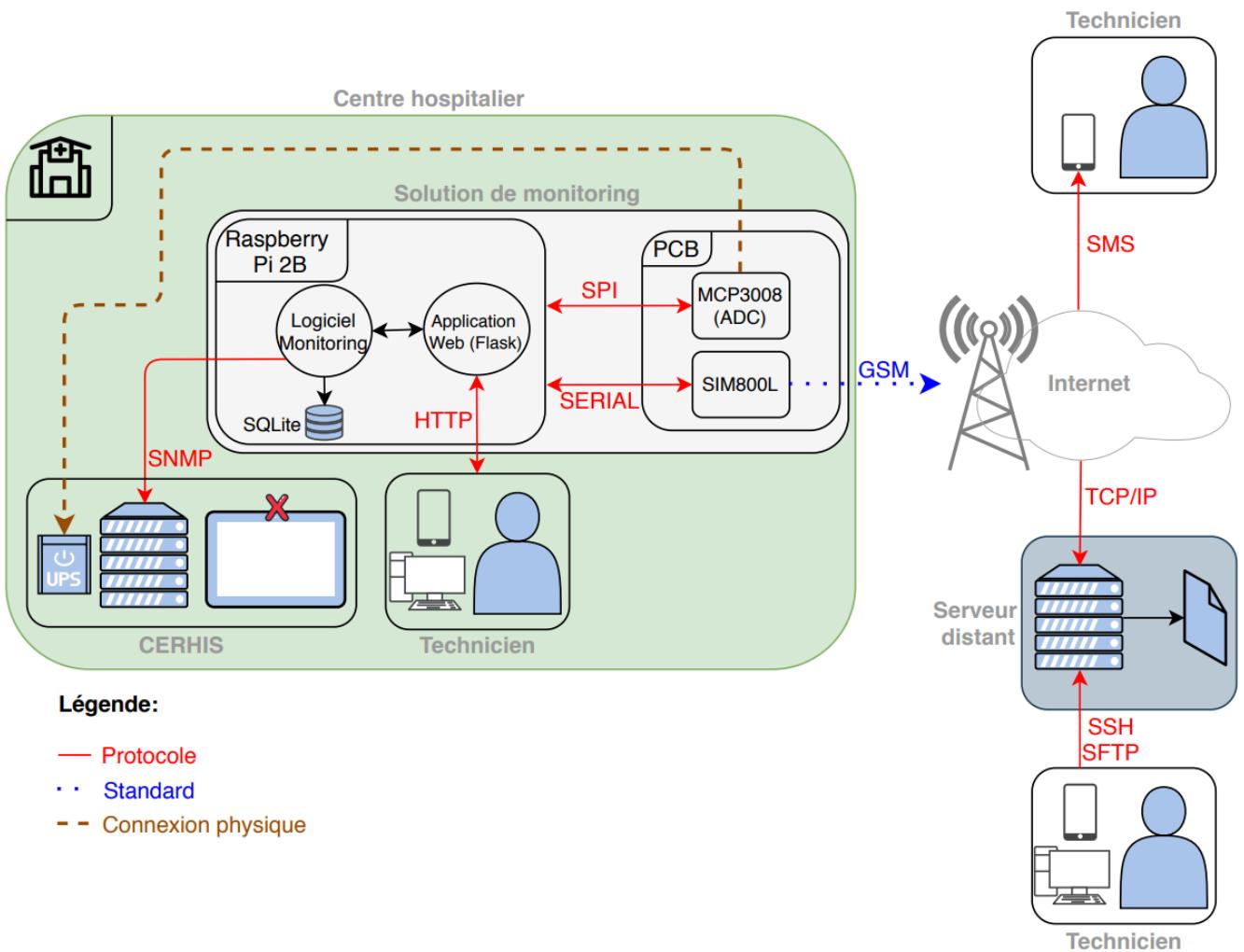


FIGURE 4.1 – Architecture de la solution de monitoring développée lors de l’année académique 2018-2019

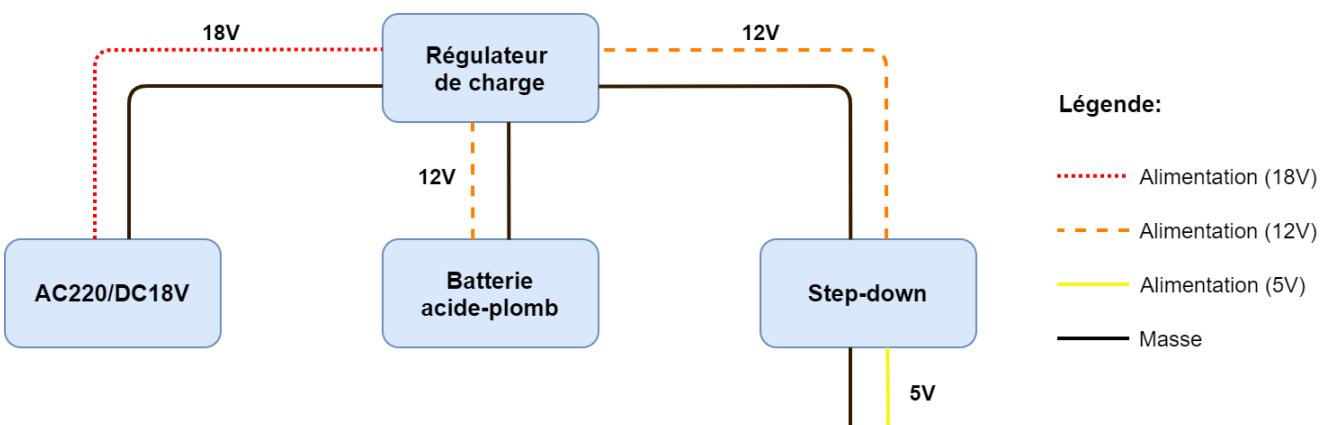


FIGURE 4.2 – Système d’alimentation de la solution de monitoring développée lors de l’année académique 2018-2019

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 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 [23], il sera à nouveau utilisé dans ce nouveau prototype. Dans l'annexe C, 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 moins 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 LoRaWAN³ 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 densément peuplées. [5, 6]

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 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

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

donner accès à certains services spéciaux, tels que vérifier le solde restant dans une carte prépayée (*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 [52]. 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. [48]
- Le protocole USSD reste disponible même lorsque la connectivité à Internet du réseau est coupée. En contrepartie, cela rendrait le service GPRS inexploitable.[35]
- La majorité des modems sur le marché, dont le SIM800L, sont compatibles avec USSD. [52]

Cependant, le protocole USSD possède aussi les désavantages suivants :

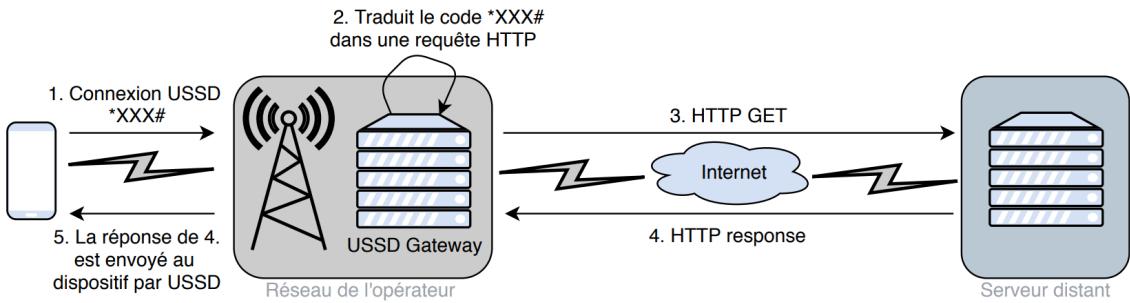
- Déployer une application utilisant USSD est très difficile puisque cela demande de travailler de très près avec un opérateur de télécommunications. En effet, toutes les connexions sont dirigées vers les serveurs de ces derniers. [58]
- Lié au problème précédent, le protocole USSD ne donne pas directement 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 [3]. 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.

Récupérer des informations :



Envoyer des informations :

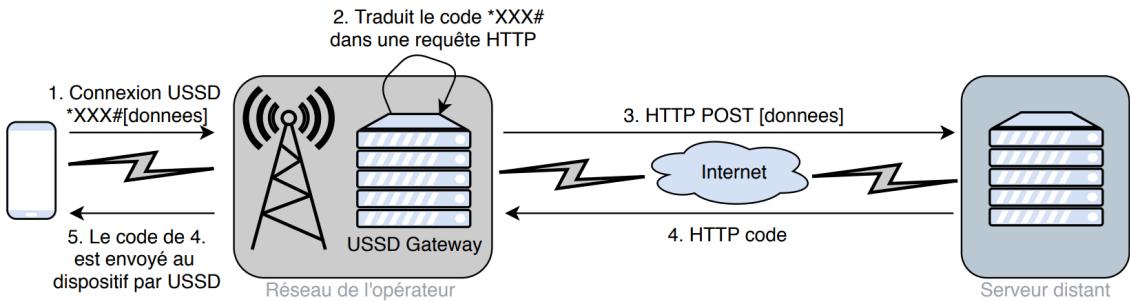


FIGURE 4.3 – Architecture simplifiée d'une solution USSD permettant d'accéder à des services sur Internet

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 WAN⁴, 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 [58]. 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

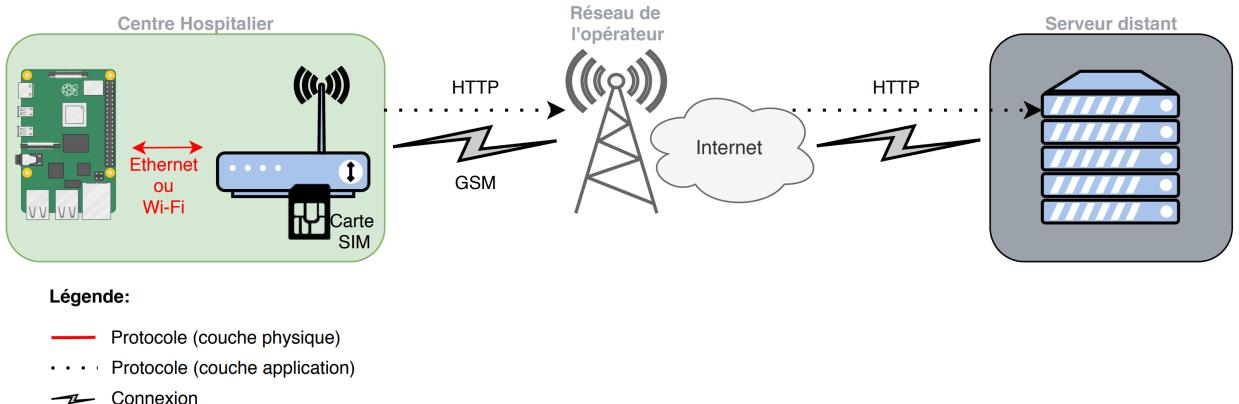


FIGURE 4.4 – Architecture simplifiée de la solution de communication avec un router LTE

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.

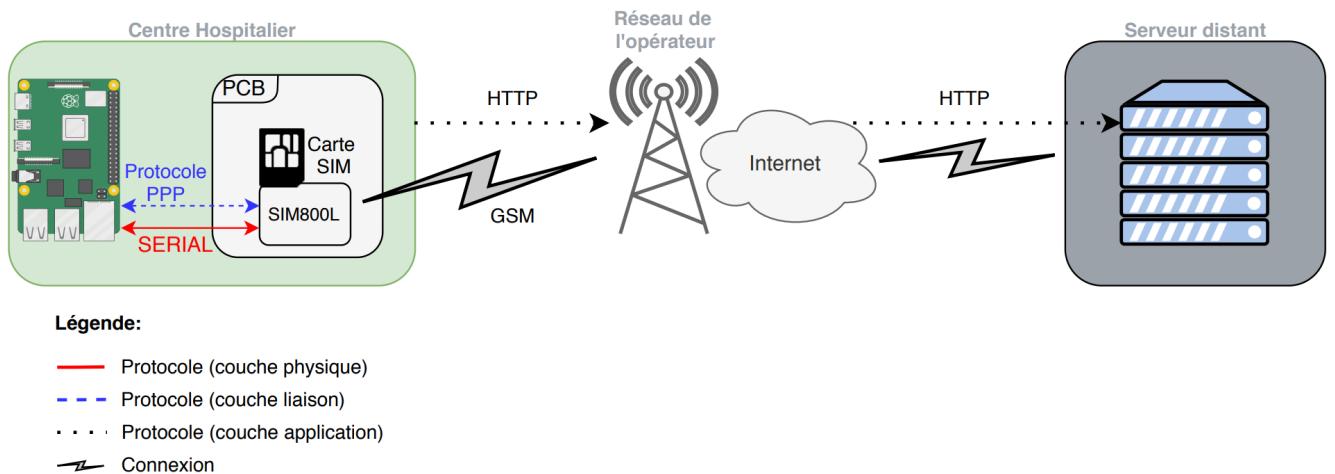


FIGURE 4.5 – Architecture simplifiée de la solution de communication avec le SIM800L et le protocole PPP

Finalement, la solution 2 (USSD) est celle qui a été implantée 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 [7]. 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.

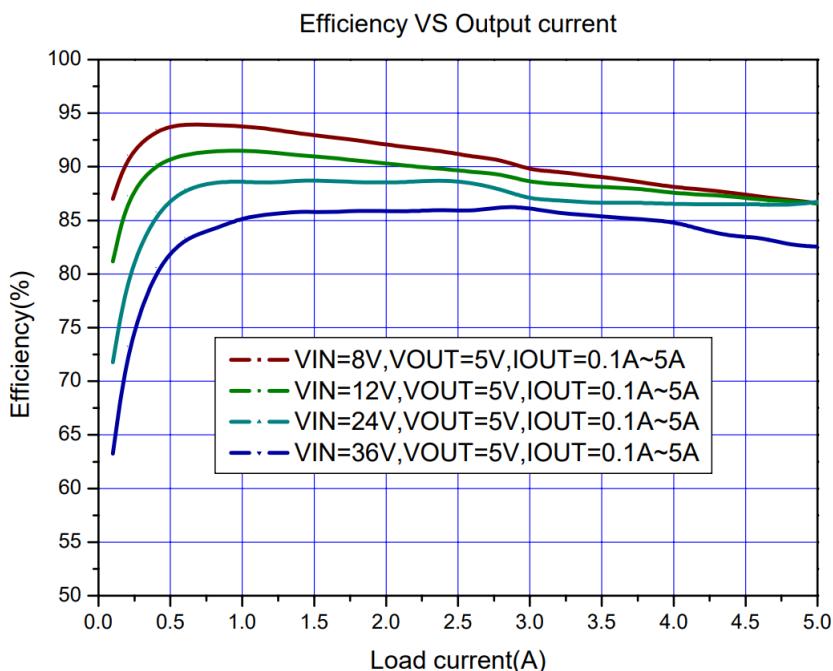


FIGURE 4.6 – Courbe de rendement du XL4015 lorsque $V_{out} = 5V$ [1]

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 MQTT⁵ pour la transmission des messages. Ce protocole a été spécifiquement conçu pour des "réseaux à faible bande passante, à haute latence et non fiables"[21]. Un SDK⁶ est fourni afin de simplifier l'accès à la plateforme de Thingstream 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 [62].

Ensuite, il y a les IP-Things. Ils correspondent aux différents clients qui se connectent au broker⁷ 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. Message Queuing Telemetry Transport

6. Software Development Kit ou kit de développement logiciel

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

- HTTP
- FTP
- SMTP (email)

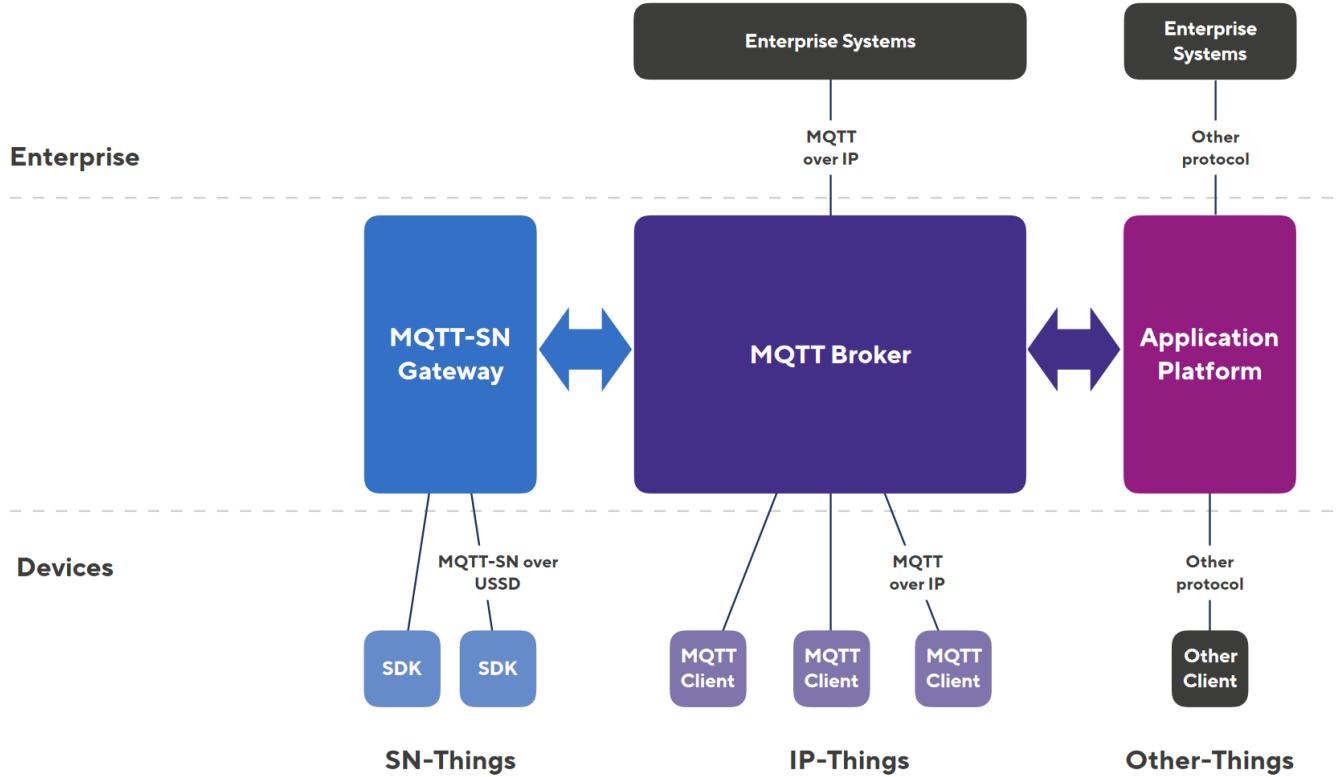


FIGURE 4.7 – Architecture simplifiée de la plateforme Thingstream [28]

Plus de détails sur cette fonctionnalité sont disponibles sur [9]. 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.

Ce protocole consiste à lancer un stress test sur deux coeurs 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.

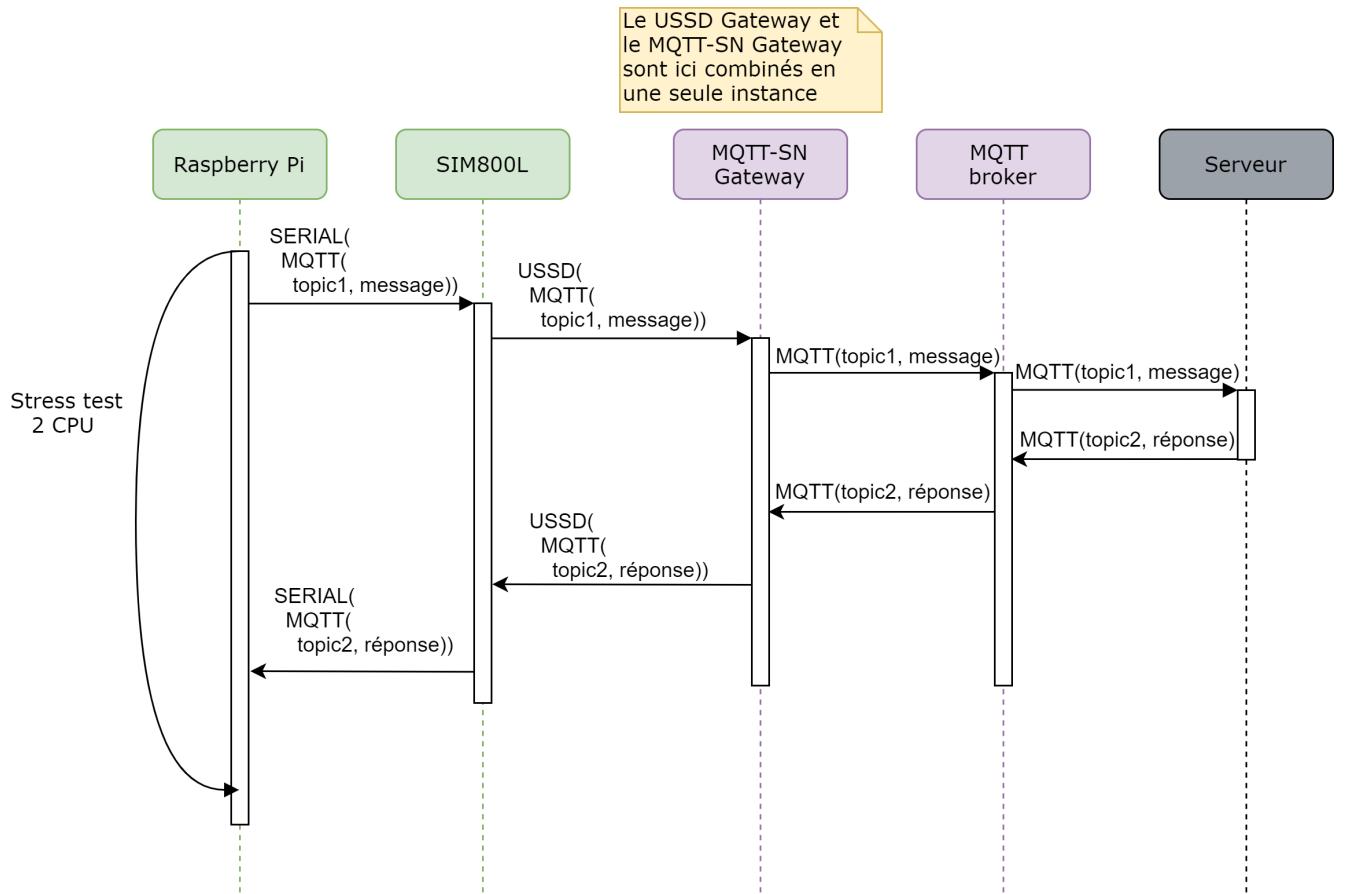


FIGURE 4.8 – Diagramme de séquence de la publication d'un message pendant un test

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 cette valeur était la même pour tous les tests. Toutefois, nous pouvons déjà conclure que le SDK

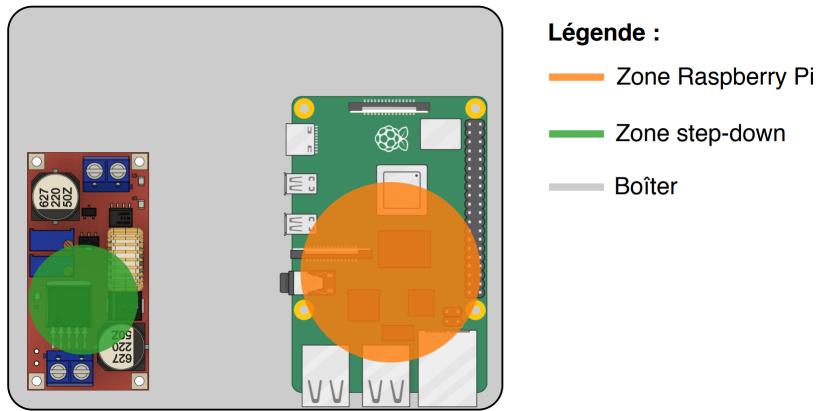


FIGURE 4.9 – Zones où la température a été mesurée lors des tests

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)

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

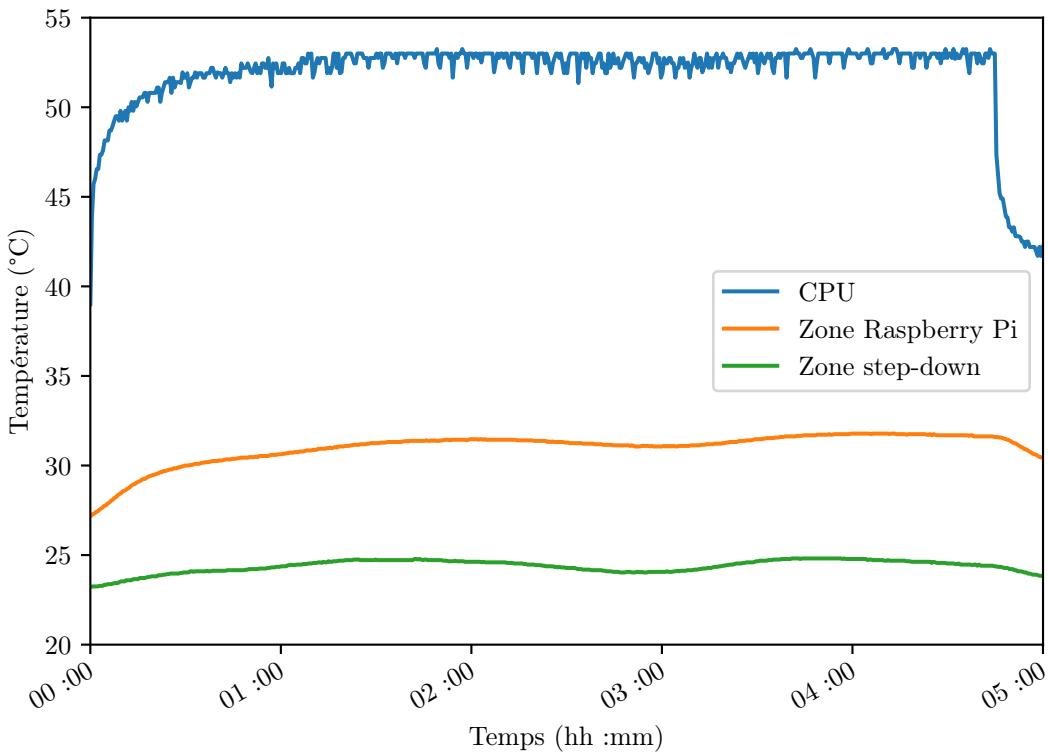


FIGURE 4.10 – **Test 1** : Évolution de la température dans les tests avec step-down et sans ventilateur

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 boîtier 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

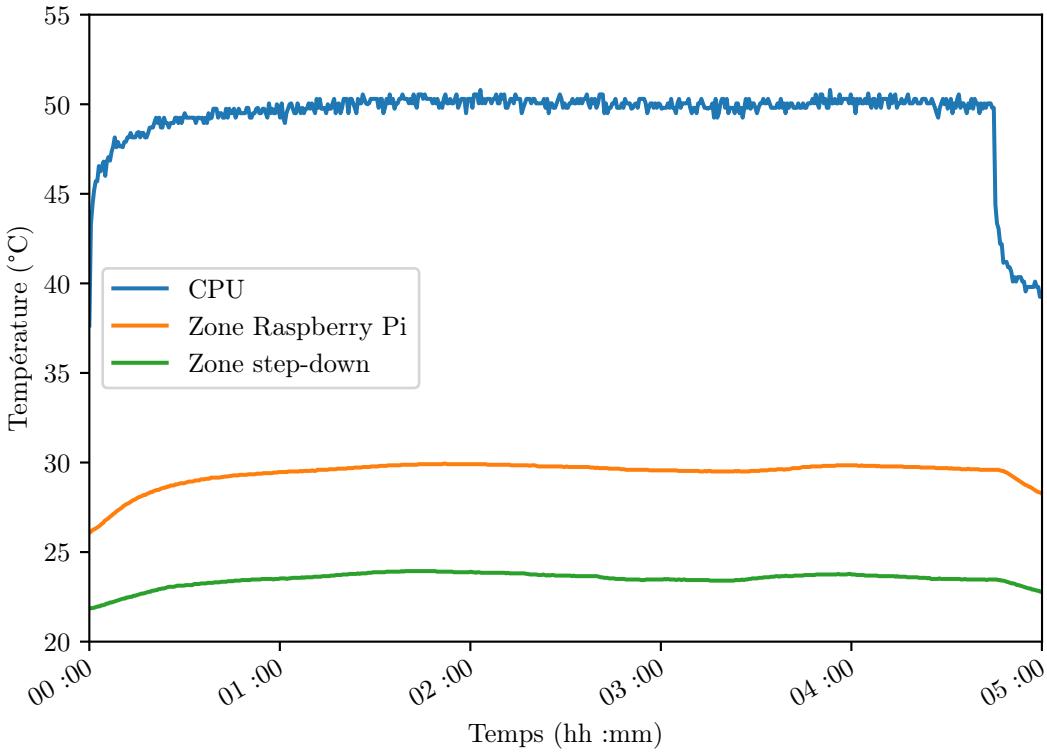


FIGURE 4.11 – **Test 2 :** Évolution de la température dans les tests sans step-down et sans ventilateur

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 l'intérieur de la boîte [32]. 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

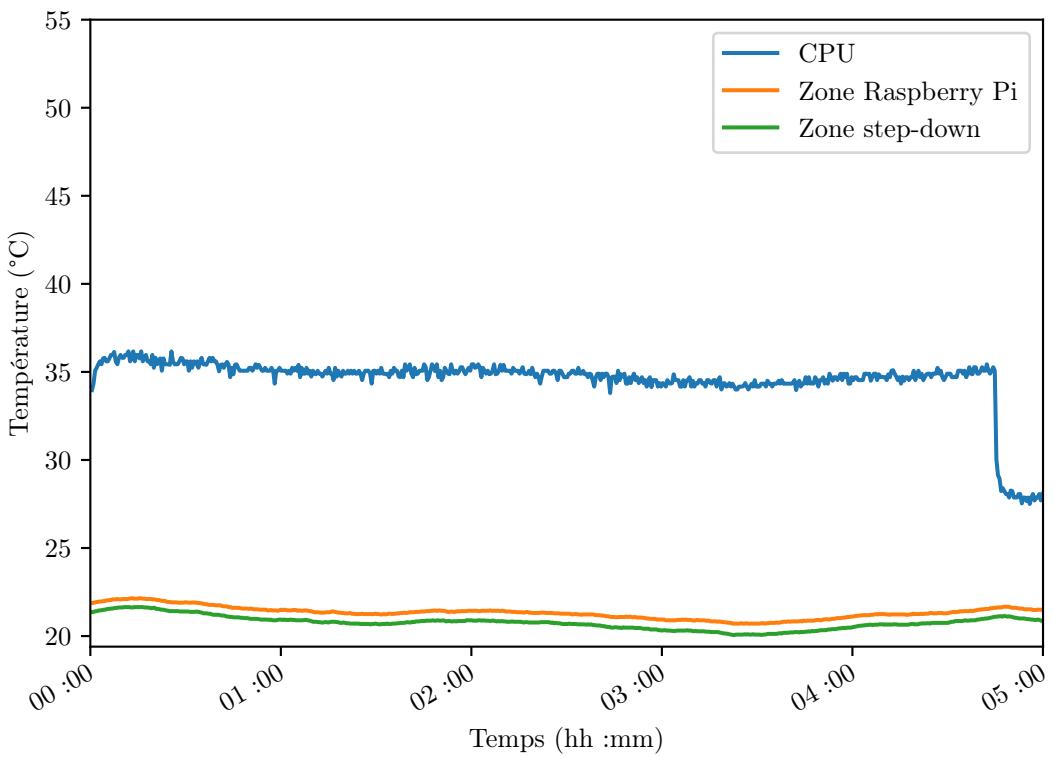


FIGURE 4.12 – **Test 3** : Évolution de la température dans les tests avec step-down et avec ventilateur

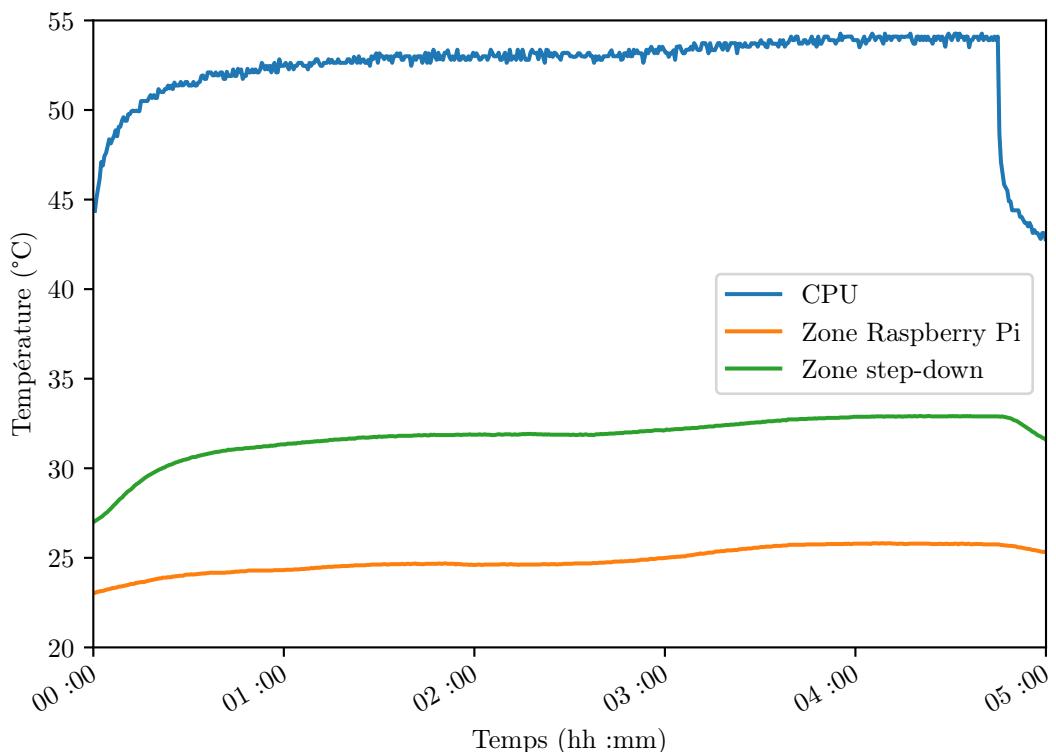


FIGURE 4.13 – **Test 4** : Évolution de la température dans les tests avec step-down et blindage magnétique mais sans ventilateur

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

TABLE 4.1 – Statistiques loggés par le SDK de Thingstream pour les 4 premiers tests



FIGURE 4.14 – Vue des faces avant et supérieure avec le couvercle

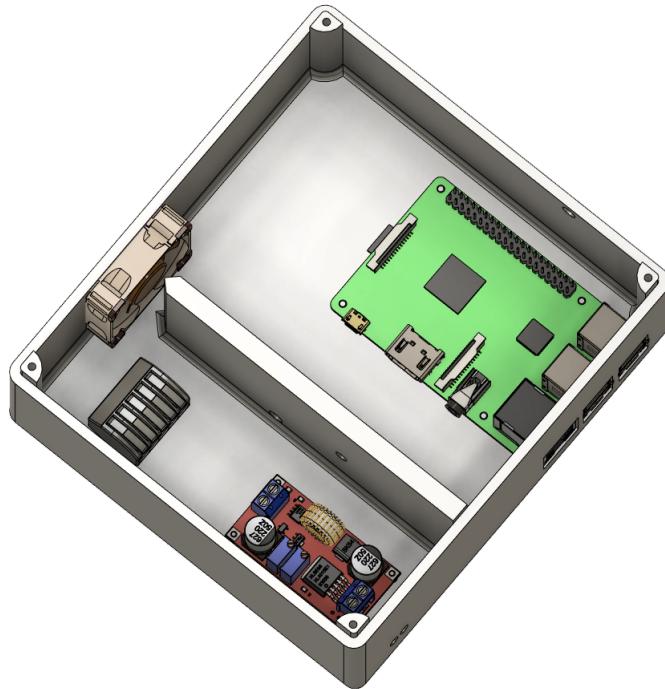


FIGURE 4.15 – Vue de la face supérieure sans le couvercle

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

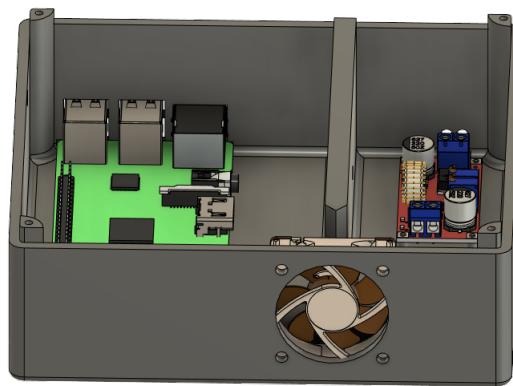


FIGURE 4.16 – Vue de la face arrière sans le couvercle

des composants est essentiel afin d'assurer le bon fonctionnement de ces derniers et de prolonger leur durée de vie.

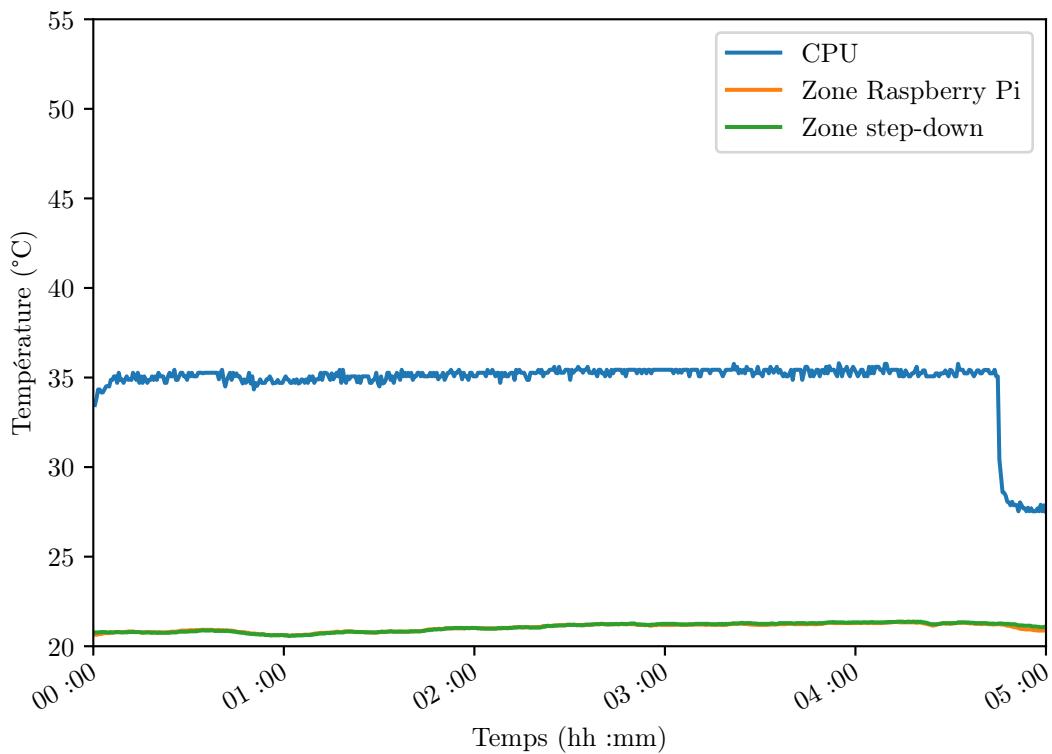


FIGURE 4.17 – **Test 5** : Évolution de la température dans les tests réalisés dans le nouveau boîtier

	Test 5	Test 6
Protocol :SendPackets Error	min : 4	min : 2
	max : 5	max : 3
	moy : 4,33	moy : 2,67
Receive :parseMessage Warning	min : 0	min : 0
	max : 0	max : 0
	moy : 0	moy : 0

TABLE 4.2 – Statistiques loggés par le SDK de Thingstream pour les tests 5 et 6

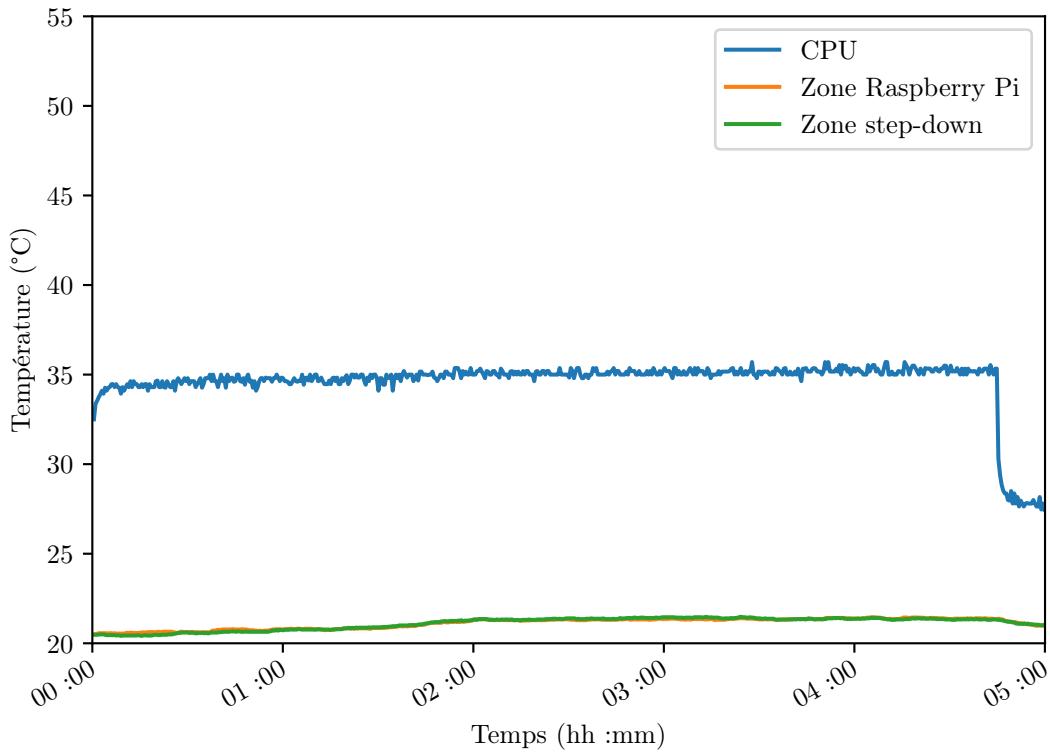


FIGURE 4.18 – **Test 6 :** Évolution de la température dans les tests réalisés dans le nouveau boîtier avec blindage magnétique

Chapitre 5

L'application

Comme expliqué plus tôt dans la section 3, l'application est divisée en deux grandes parties :

- La solution de monitoring locale tournant sur le Raspberry Pi. Ce logiciel est chargé de collecter toutes les informations sur les systèmes à moniter et de les transmettre au serveur distant. Elle doit également inclure une interface graphique accessible localement permettant de visualiser toutes les données recueillies.
- Le serveur distant qui reçoit et stocke toutes les données envoyées par le logiciel de monitoring. De plus, tout comme ce dernier, le serveur doit aussi exposer ces données à travers une interface graphique.

Ceci permet aux techniciens de soit vérifier l'état de CERHIS en se connectant au serveur distant via Internet, ou soit ils peuvent le vérifier localement en accédant à l'interface proposée par le service de monitoring. Cette dernière interface est particulièrement intéressante si aucune connexion Internet n'est disponible dans le centre hospitalier.

La solution de monitoring sera présentée dans la section 5.1, suivie par la description de l'implémentation du serveur distant dans la section 5.2.

5.1 Client

5.1.1 Architecture

Avant de démarrer l'implémentation de tout logiciel, il faut d'abord commencer par la conception de son architecture. Dans ce projet, plusieurs facteurs doivent être pris en compte lors de l'étape de conception. Premièrement, ce mémoire est destiné à l'aide au développement. De ce fait, il est tout à fait primordial que le code soit facilement accessible par d'autres personnes qui n'ont pas été impliquées dans le développement de celui-ci. De plus, ces mêmes personnes doivent pouvoir continuer à utiliser ce code et à ajouter de nouvelles fonctionnalités dessus.

Ensuite, comme expliquée plus tôt, l'utilisation de Thingstream n'est pas sans risque. Le tableau 5.1 reprend la majorité des risques liés à l'utilisation de Thingstream et la probabilité qu'ils aient lieu. Pour la plupart des risques, il est peu probable qu'ils provoquent un incident à l'avenir, ou que l'impact soit aussi conséquent qu'un arrêt total du service. Toutefois, atténuer l'impact de ces risques est très important et cela peut être réalisé grâce à l'architecture du logiciel.

Risques	Probabilité	Justification
Fin de la commercialisation de Thingstream	Peu Probable	Thingstream a été acquis par u-blox récemment
Augmentation du coût d'utilisation de Thingstream	Très Probable	Cela est déjà arrivé cette année, mais similaire aux autres opérateurs
Diminution de la qualité du service	Peu Probable	Thingstream perdrait la majorité de ses clients
Bugs dans le SDK	Probable	Comme tout logiciel informatique. L'impact dépendra du temps requis par Thingstream pour le corriger

TABLE 5.1 – Risques liés à l'utilisation de Thingstream

Pour mitiger les risques liés à Thingstream, toute la partie de communication est encapsulée dans une application unique qui n'est nullement attachée à la solution de monitoring. L'application de communication dispose d'un socket serveur TCP/IP qui autorise tout autre logiciel de s'y connecter pour échanger des messages. Ces messages ont un format spécifique permettant au système de communication d'interpréter exactement comment les traiter. De plus amples détails sur l'implémentation de cette solution seront fournis dans la section suivante. Toutefois, après ce premier élément, l'architecture du logiciel est celle présentée dans la figure 5.1. Si jamais le service de Thingstream n'est plus commercialisé, un serveur distant peut prendre le rôle de l'application de communication. Le logiciel de monitoring peut alors continuer à utiliser le protocole TCP/IP pour échanger des informations (voir figure 5.2). De ce fait, une connexion Internet peut remplacer Thingstream sans apporter de modifications majeures au logiciel de monitoring.

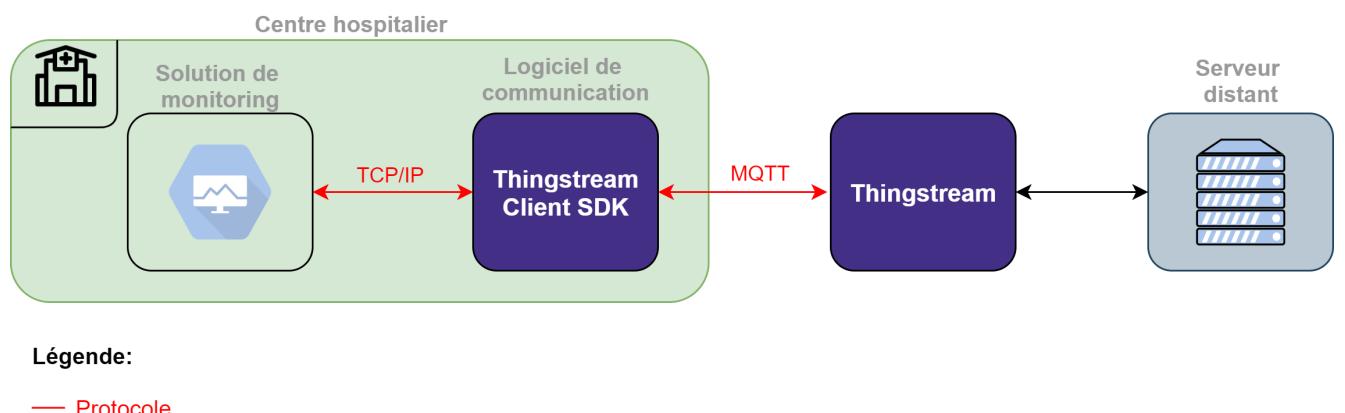
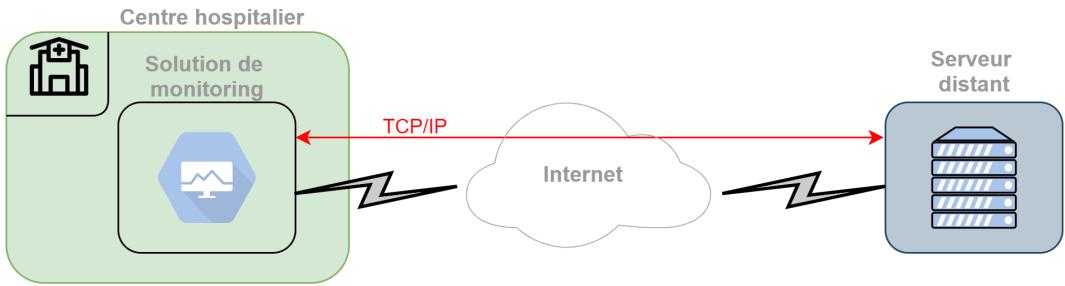


FIGURE 5.1 – Architecture après l'ajout du logiciel de communication

Ensuite il y a l'architecture de toute la solution de monitoring. Cette part comprend plusieurs fonctionnalités d'acquisition de données qui ne sont pas forcément liées entre elles. Par exemple, il y a l'acquisition de données concernant l'état du serveur ou la présence des tablettes. Cependant, certaines parties doivent pouvoir communiquer entre elles, comme le service de présence des tablettes et la fonction de mappage du réseau. En effet, le service de présence doit d'abord connaître quels sont les dispositifs à montrer et cela est accompli en regardant le résultat du mappage.



Légende:

— Protocole

FIGURE 5.2 – Architecture simplifiée de la solution de monitoring sans Thingstream

Généralement, jusqu'à il y a quelques années, les applications avaient une architecture monolithique et des patrons de conception étaient utilisés pour séparer les diverses parties du logiciel. Au cours des dernières années, une nouvelle architecture a fait face, les microservices. Cette conception se met en avant par le fait qu'elle brise les différents composants d'une application monolithique en plusieurs mini applications, aussi appelés les microservices. Ces services peuvent ensuite communiquer entre eux grâce à des protocoles distincts tels que HTTP, RPC¹ et RabbitMQ. La figure 1 reprend la différence entre les deux architectures.

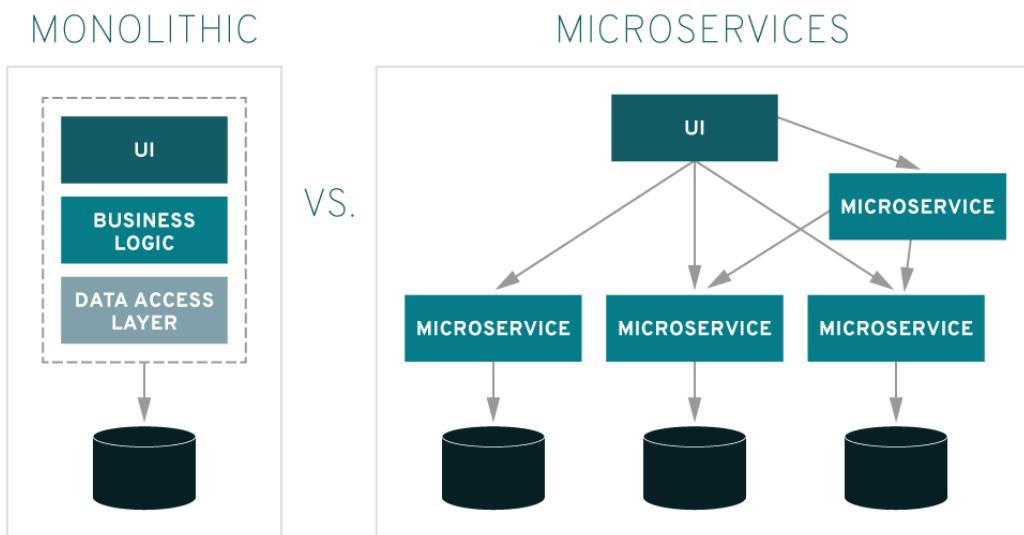


FIGURE 5.3 – Différence entre les architectures monolithique et microservices [24]

Comparée à une architecture monolithique, les microservices possèdent les avantages suivants [24] :

- Mise sur le marché plus rapide
- Haute évolutivité
- Résilience
- Facilité de déploiement
- Accessibilité
- Ouverture

1. Remote procedure call

Tous ces avantages sont particulièrement intéressants, particulièrement l'accessibilité, la haute évolutivité et la résilience. Ces trois grands avantages vont ensemble avec ce qui est requis pour ce projet. De ce fait, cette architecture est celle qui a été implémentée lors de ce mémoire. La section suivante couvrira chacun de ces microservices et comment ils interagissent entre eux.

Cette section commence par la description de l'implémentation de chaque service. Ensuite, il sera question de la manière dont les services interagissent. Premièrement, les services en charge de la communication seront présentés puisque ce sont ceux qui ont été développés en premier. En effet, le choix d'utiliser Thingstream a été fait avant de considérer les solutions de monitoring possibles. Compte tenu des problèmes rencontrés avec la solution de communication l'année précédente, commencer par le développement de cette solution semblait être le choix plus judicieux. Cela signifie que la solution de monitoring s'adapte aux capacités de la solution de communication, et non l'inverse.

5.1.2 Implémentation des fonctionnalités

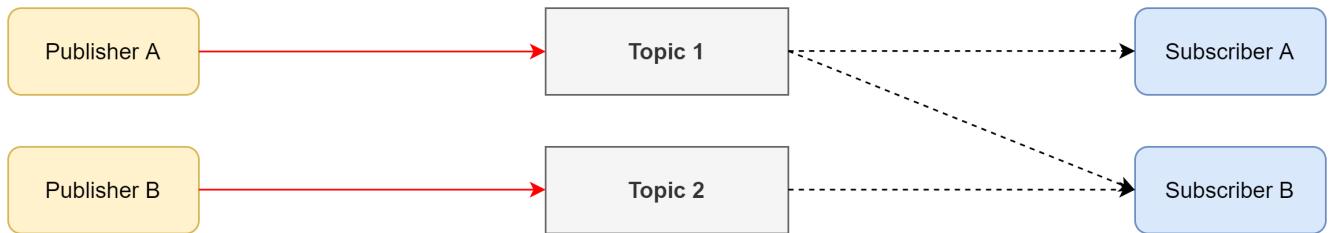
Envoi des données

Comme expliqué dans la section 4.2.2, Thingstream propose un SDK qui permet de très aisément avoir accès la plateforme et de l'exploiter. Ce SDK gère toute la communication de bas niveau avec le modem, le SIM800L en l'occurrence. Son utilisation est très similaire à celle de toutes les autres bibliothèques logicielles de clients MQTT qui sont actuellement disponibles. Cette partie ne couvre donc que la façon dont un message est géré au niveau du protocole MQTT. À partir de cet instant, c'est le SDK de Thingstream qui s'en occupe et n'est donc pas discuté ici. Veuillez aussi noter que le choix de Thingstream a été effectué avant de démarrer le développement du logiciel de monitoring. Comme expliqué précédemment, la partie de communication a été très problématique lors du dernier projet. De ce fait, la décision fut d'adapter le logiciel à la solution de communication et non l'inverse étant donné la complexité de cette dernière.

Dans le but de mieux comprendre le fonctionnement de la solution de communication, il faut d'abord découvrir les bases du protocole MQTT. Comme expliqué précédemment, ce protocole a été spécifiquement conçu pour être utilisé dans des réseaux non fiables. L'architecture du protocole MQTT est basée sur le patron de communication publish-subscribe. Des *publishers*(éditeurs) publient des messages sur un topic et les *subscribers*(abonnés) reçoivent tous les messages qui ont été publiés sur le topic auquel ils sont abonnés. La figure 5.4 reprend un exemple de ce patron de communication. Deux *publishers* publient un message sur deux topics différents. Le *subscriber* A est seulement abonné au topic 1 et, par conséquent, ne reçoit que le message publié par le *publisher* A. Cependant, le *subscriber* B reçoit tous les messages puisqu'il est abonné aux deux topics. Les messages envoyés ne possèdent donc pas de destinataire en particulier. De plus, les *publishers* ne savent pas a priori qui lira les messages qu'ils ont publiés.

Cette méthode permet de découpler les *publishers* et les *subscribers*. En effet, ils ne communiquent pas directement entre eux, mais font appel à une tierce partie, le *broker*, pour échanger les messages. Au moment de la connexion, les *subscribers* envoient au *broker* la liste de topics auxquels ils veulent s'abonner. Lorsqu'un *publisher* publie sur un topic, le *broker* identifie tous les *subscribers* souscrits à ce topic et leur transmet le message. La figure 5.5 présente un diagramme de séquence de ce procès.

Ceci constitue l'architecture de base du protocole MQTT. Cependant, les clients peuvent détenir les rôles de *publisher* et de *subscriber* à la fois, ils ne sont donc pas limités à une seule fonction. Un autre point important à comprendre à propos MQTT est la façon dont les topics fonctionnent. Un



Légende:

— Publish / Publier

- - - Subscribe/ Abonné

FIGURE 5.4 – Patron de conception Publish-Subscribe

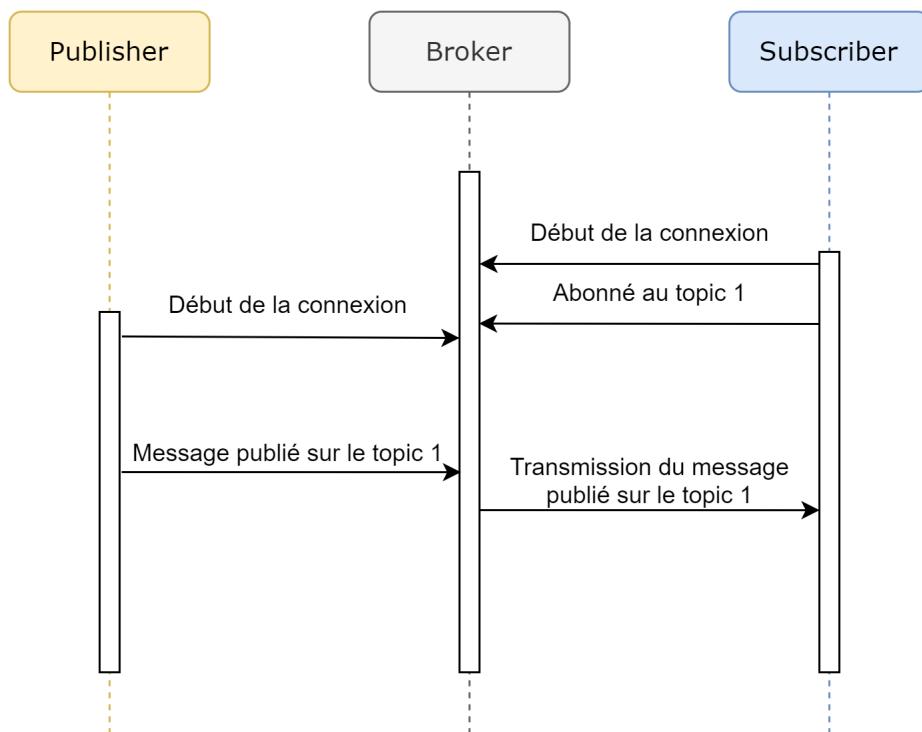


FIGURE 5.5 – Diagramme de séquence affichant un échange de données suivant le patron de conception Publish-Subscribe

topic est représenté par une chaîne de caractères et il peut avoir plusieurs niveaux. Un caractère barre oblique (/) sépare chaque niveau. Par exemple, imaginons qu'un capteur de température se trouve dans le salon au rez-de-chaussée. La figure 5.6 montre un exemple de topic qui pourrait être exploité par ce capteur pour transmettre ces données à travers le protocole MQTT. Cependant, supposons maintenant qu'un client veut s'abonner à tous les capteurs de température présents au rez-de-chaussée. Évidemment, ce client pourrait souscrire à chaque capteur individuellement. Pour éviter ce genre de situation, MQTT permet d'employer des wildcards pour qu'un client puisse s'abonner à plusieurs topics qui possèdent une structure similaire. Les figures 5.7 et 5.8 affichent la façon dont les wildcards peuvent être appliqués. Dans le logiciel de communication de ce mémoire, seulement le wildcard à niveau unique est utilisé.



FIGURE 5.6 – Exemple de topic MQTT [33]



FIGURE 5.7 – Exemple d'utilisation du wildcard à niveau unique pour matcher plusieurs topics [33]

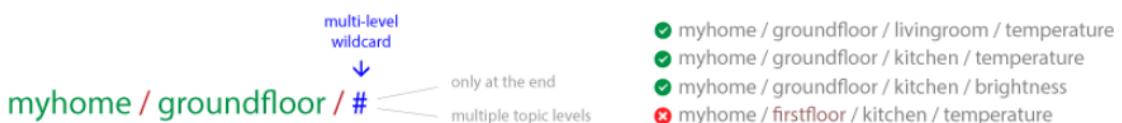


FIGURE 5.8 – Exemple d'utilisation du wildcard de plusieurs niveaux pour matcher plusieurs topics [33]

Le protocole MQTT possède encore d'autres fonctionnalités comme la qualité de service. Toutefois, ces fonctionnalités supplémentaires n'ont pas été exploitées lors de ce mémoire et ne sont donc pas spécifiées ici. Veuillez aussi noter que le client décrit dans cette section emploie MQTT-SN, et non pas la version standard de MQTT. En effet, ce client est ce que Thingstream appelle un SN-Thing. Cela dit, le fonctionnement des deux protocoles est très similaire. Contrairement à MQTT, MQTT-SN utilise des alias pour remplacer le nom complet des topics et permet aux clients de se déconnecter temporairement dans le but d'économiser de l'énergie. Les messages qui devraient être envoyés au client sont alors sauvegardés dans le broker jusqu'à la reconnexion du client. Les sources proposent des informations plus détaillées sur MQTT et MQTT-SN. La dernière spécification complète de MQTT peut être trouvée dans [34].

Le client de communication développé exploite trois topics afin de transmettre des informations. Tous les messages sont encodés dans le format de données JSON², indépendamment du topic. Ce format est facilement lisible par un humain et est supporté par la majorité des langages de programmation. En outre, son utilisation est très répandue sur le web. Cependant, selon le topic, la structure du JSON est différente. Ci-dessous est présentée la liste des trois topics et le format de données utilisé dans chacun d'eux³ :

- **/device/#identity#/startup** : Le client utilise ce topic lorsque l'application exploitant le client de communication doit s'authentifier auprès du serveur. Les messages publiés sur ce topic ont la forme suivante :

```

1 {  
2   "id" : #id#,  
3   "location" : #localisation_dispositif#  
4 }
```

2. JavaScript Object Notation

3. #identity# est un ID unique attribué par Thingstream à chaque client

- **/device/#identity#/post** : Ce topic est utilisé afin de transmettre les données envoyées par l'application au serveur. Un en-tête doit être ajouté à tous les messages publiés sur ce topic afin d'identifier le type des données publiés. La structure des messages est la suivante :

```

1 {
2   "type" : #type_contenu#,
3   "content" : [ #donnees# ]
4 }
```

- **/device/all/sms** : Ce topic est utilisé pour envoyer des SMS au techniciens. Contrairement aux deux topics précédents, le serveur n'est pas souscrit à ce topic. En effet, c'est un Data Flow de Thingstream qui y est abonné. Tous les messages publiés sur ce topic sont convertis en un SMS qui est envoyé à un destinataire. De ce fait, les champs du JSON sont différents de ceux du topic post. Le Data Flow est affiché dans la figure 5.9 et la structure d'un message publié sur ce topic est la suivante :

```

1 {
2   "message" : #contenu_sms#,
3   "to" : #numero_gsm_recipient#
4 }
```

✖ Sending SMS

Test: Version 1 Subscribed to device/all/sms

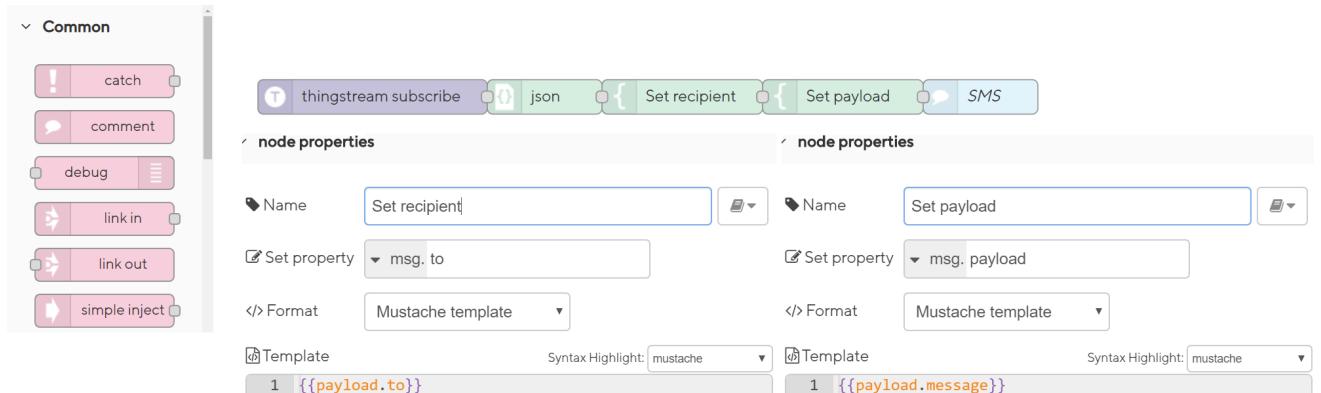


FIGURE 5.9 – Exemple de Data Flow de Thingstream permettant d'envoyer un SMS

En plus des trois topics sur lesquels le client publie des messages, le client s'abonne à un quatrième topic afin que le serveur puisse lui adresser des messages. Ce topic est **device/#identity#/recv** et sera expliqué plus en détail dans la section 5.2. Toutes les informations reçues par le client de communication sur ce topic sont transmises au logiciel de monitoring.

Comme défini dans l'architecture du programme, l'application de communication est complètement isolée et accepte les connexions sur un socket de serveur TCP/IP. Par conséquent, la solution de monitoring ne possède aucune information sur le fonctionnement du logiciel de communication décrit ci-dessus. Il ne fait qu'envoyer les données sur le socket et le laisse le client de communication se charger du reste. Cependant, elles possèdent deux destinataires possibles, le serveur distant ou

le récipiendaire des SMS. Dans le but de créer une distinction entre ces deux types de messages, ils sont encapsulés par un autre JSON qui a la forme suivante :

```

1 {
2   "type" : "server" ou "sms",
3   "payload" : #numero_gsm_recipient#
4 }
```

Cette capsule est lue par le client de communication qui décide ensuite sur quel topic il faut publier le message, mais seulement le contenu du champ *payload* est publié.

Un dernier aspect important de ce client de communication est qu'il permet à une seule application de se connecter à la fois au serveur distant. Créer un client capable de gérer des connexions augmenterait considérablement sa complexité sans apporter aucun avantage au projet. Mais, puisque le client n'accepte qu'une seule connexion, il doit s'assurer que c'est bien celle du logiciel de monitoring qui est approuvé. C'est pourquoi un protocole a été mis en place qui permet d'authentifier l'application auprès du serveur. Le diagramme de séquence de la figure 5.11 présente le déroulement du protocole.

Du côté de l'application de monitoring, un service unique reçoit toutes les données et se charge de les transmettre au client de communication. Avant la transmission, il formate les données en fonction du destinataire et du type de données. Le fonctionnement exact de ce service sera détaillé au cours des sections suivantes et il sera identifié par le nom *sender service*. La figure 5.10 présente l'architecture complète après l'introduction du *sender service* et le service de communication.

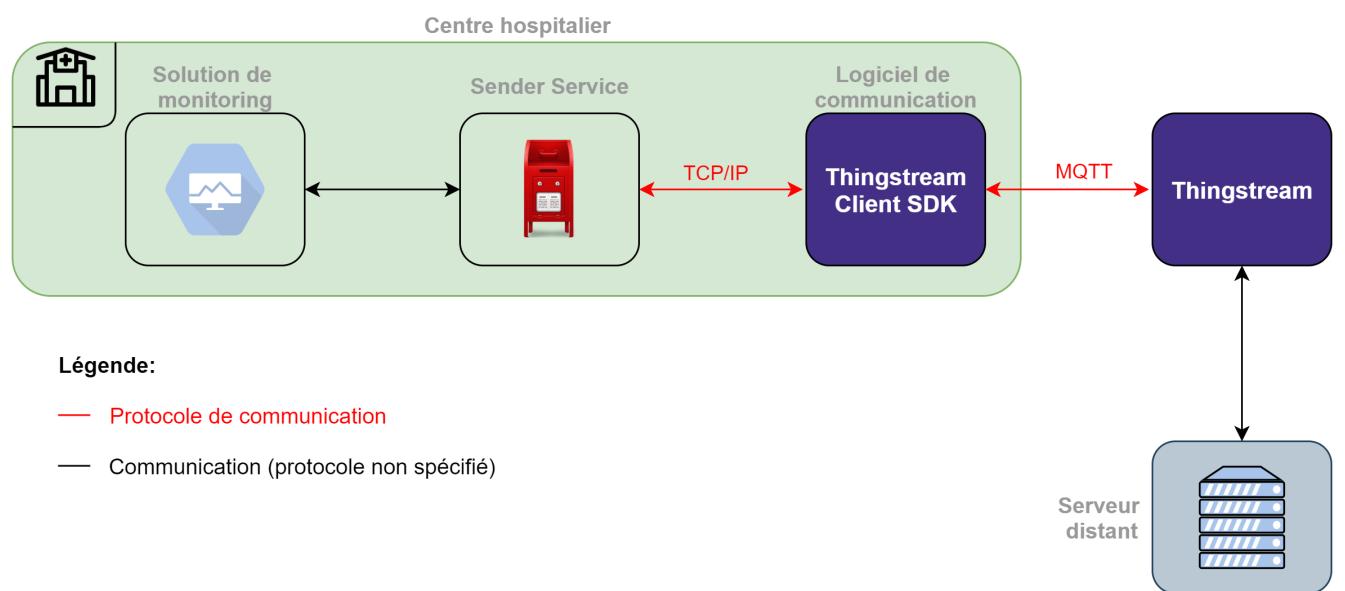


FIGURE 5.10 – Architecture après l'ajout du *sender service*

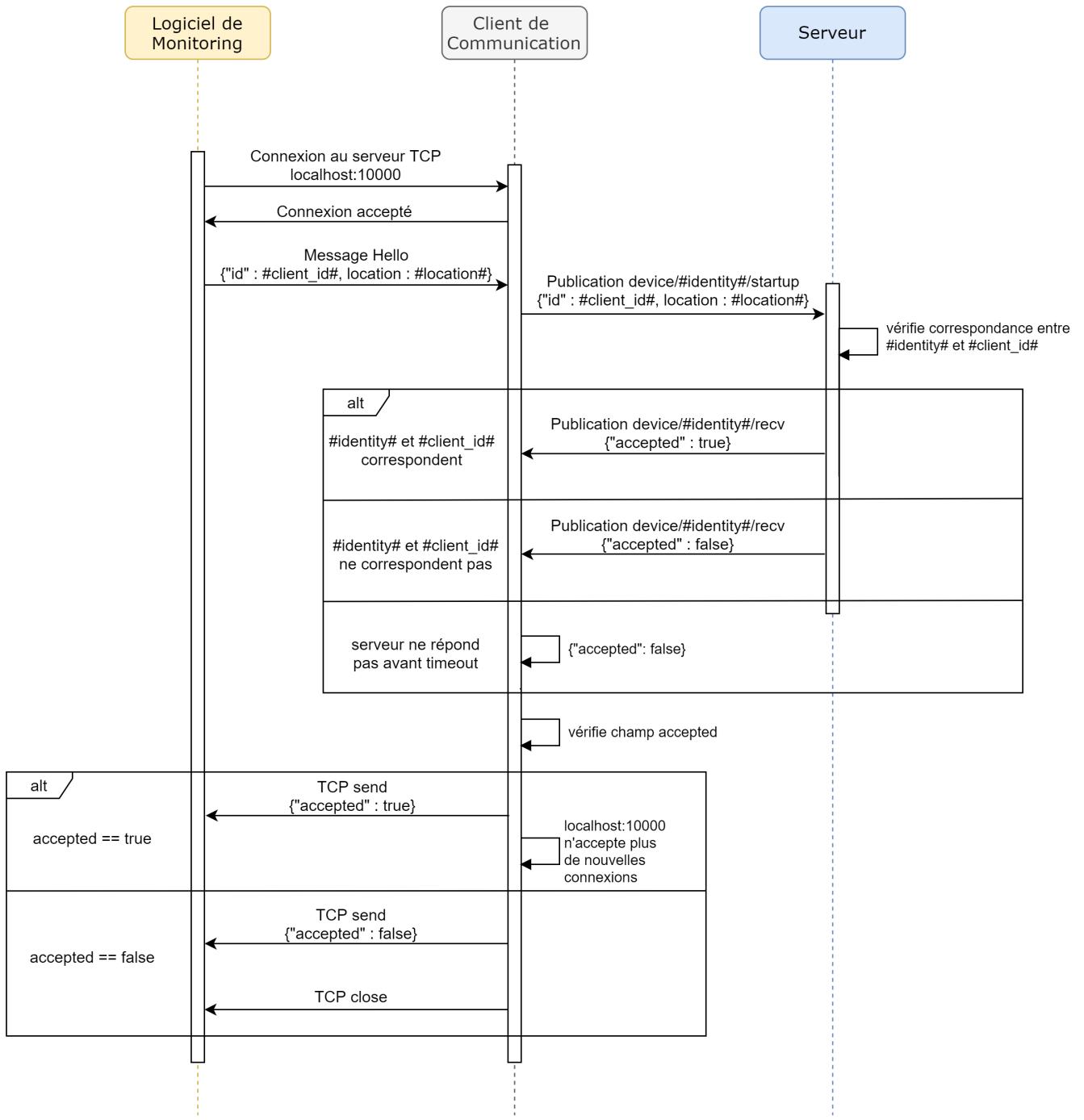


FIGURE 5.11 – Diagramme de séquence à propos le processus de connexion au serveur distant

Monitoring du serveur

Dans l'intention de surveiller l'état du serveur, le protocole SNMP⁴ a été utilisé lors des projets antérieurs. Ce protocole a été conçu spécifiquement pour surveiller, diagnostiquer et administrer des dispositifs. Presque tous les types d'appareils connectés à un réseau IP sont pris en charge, y compris les serveurs, les imprimantes et les routeurs. Puisque le cahier des charges ne stipule qu'un nombre réduit de métriques du serveur à surveiller, le protocole SNMP s'est montré comme une solution simple à implémenter. Bien entendu, étant une solution existante depuis un bon nombre d'années, la majorité des langages de programmation propose des bibliothèques logicielles pour

4. Simple Network Management Protocol

exploiter SNMP.

Lors de ce mémoire, une première implémentation utilisant SNMP pour récupérer les valeurs du serveur a été réalisée. Ce logiciel était capable de retrouver toutes les valeurs précisées dans le cahier des charges sans présenter aucun problème. Cependant, cette solution manquait d'évolutivité. En effet, une quantité de code considérable était déjà requise pour récupérer, traiter et sauvegarder l'*uptime* du serveur ainsi que l'état des processus. En outre, les parties prenantes du projet avaient exprimé le souhait de disposer d'une application hautement configurable au niveau de la définition des alertes. Ce dernier aspect augmentait encore plus la complexité du système.

Trouver une meilleure solution est alors devenu primordial. Pour ce faire, les différents logiciels de monitoring disponibles sur le marché ont été étudiés. Quelques critères ont été établis en vue d'identifier celui qui est le plus adapté à ce projet. Premièrement, la solution devrait être entièrement gratuite. Plusieurs solutions payantes existent, mais leur coût est très important. Deuxièmement, le logiciel devrait être facile à utiliser et présenter une bonne et ample documentation. Ensuite, il devrait être hautement configurable, permettant de l'ajuster aux besoins de CERHIS. Finalement, le logiciel devrait être compatible avec le plus grand nombre possible d'appareils de CERHIS.

Après avoir posé ces critères, trois solutions sont ressorties :

- **Prometheus** est un logiciel open source de monitoring et *alerting toolkit* qui est actuellement maintenu par la Cloud Native Computing Foundation⁵. Prometheus est basé sur le modèle *pull*, mais offre aussi la possibilité de l'utiliser avec le modèle *push*. Quelques fonctionnalités très intéressantes proposées par ce logiciel sont la base de données orientée séries temporelles, un modèle de données multidimensionnelles et la compatibilité avec Grafana. Prometheus est une solution très récente, créée seulement en 2012 et passée sous la responsabilité de la CNCF en 2016. De ce fait, son paradigme est différent des applications de monitoring traditionnelles. Prometheus se concentre sur les services [38] contrairement aux applications classiques qui ont principalement la machine en tête.
- **Zabbix** est aussi un logiciel open source existant depuis 2001 et conçu pour le monitoring des divers dispositifs IT. Son architecture est basée sur le modèle *push*. Zabbix exploite un large spectre de protocoles afin de surveiller une infrastructure IT. En outre, des agents sont également proposés pour une grande variété de plateformes, notamment des agents pour les systèmes d'exploitation Linux et Android. Par rapport à Prometheus, cette solution est un peu plus traditionnelle, même si elle reçoit des mises à jour assez régulièrement. Une interface graphique permettant de configurer le monitoring et de visualiser les données est incluse.
- **PandoraFMS Community**, tout comme les deux autres propositions, est aussi un logiciel open source. Cependant, cette solution possède également une version payante avec des fonctionnalités supplémentaires. Étant une solution disponible sur le marché depuis 2004, cette solution est très similaire à Zabbix. Ces deux plateformes possèdent quelques fonctionnalités exclusives, mais elles ne sont pas intéressantes dans le cadre de ce projet. D'un point de vue du logiciel, Pandora est la solution qui a besoin de plus de ressources.

Pour finir, le choix s'est porté sur Prometheus. PandoraFMS a été éliminé assez rapidement, car cette solution est plus gourmande en ressources. De plus, l'image PandoraFMS pour le Raspberry Pi semble être payante. (La documentation n'est pas super claire sur cet aspect) L'agent Android

5. CNCF

proposé par Zabbix était le plus grand atout de cette solution. Cependant, certains de ces agents sont payants et ils sont tous proposés par de tierces parties. En outre, ils ne sont pas mis à jour très souvent et leur documentation est très limitée.

Prometheus ressort donc comme étant le choix le plus sûr. Tous les clients *exporters* disponibles pour cette solution sont open source, mais créer des clients supplémentaires est également très facile. En effet, une bibliothèque logicielle est disponible en plusieurs langages de programmation, ce qui rend l'implémentation très facile. De plus, Prometheus peut également être utilisé comme une simple base de données orientée séries temporelles. Cela permet donc de sauvegarder des valeurs comme la tension de la batterie de CERHIS sans devoir recourir à d'autres bases de données.

L'architecture de Prometheus est présentée sur la figure 5.12. Comme expliqué précédemment, Prometheus *pull* des métriques au niveau des différents services. Ces services peuvent exporter des valeurs liées à l'état d'une machine, d'une application, d'une base de données ou de n'importe quel autre composant pouvant être surveillé par des métriques. Afin de *pull* les données, Prometheus utilise le protocole HTTP.

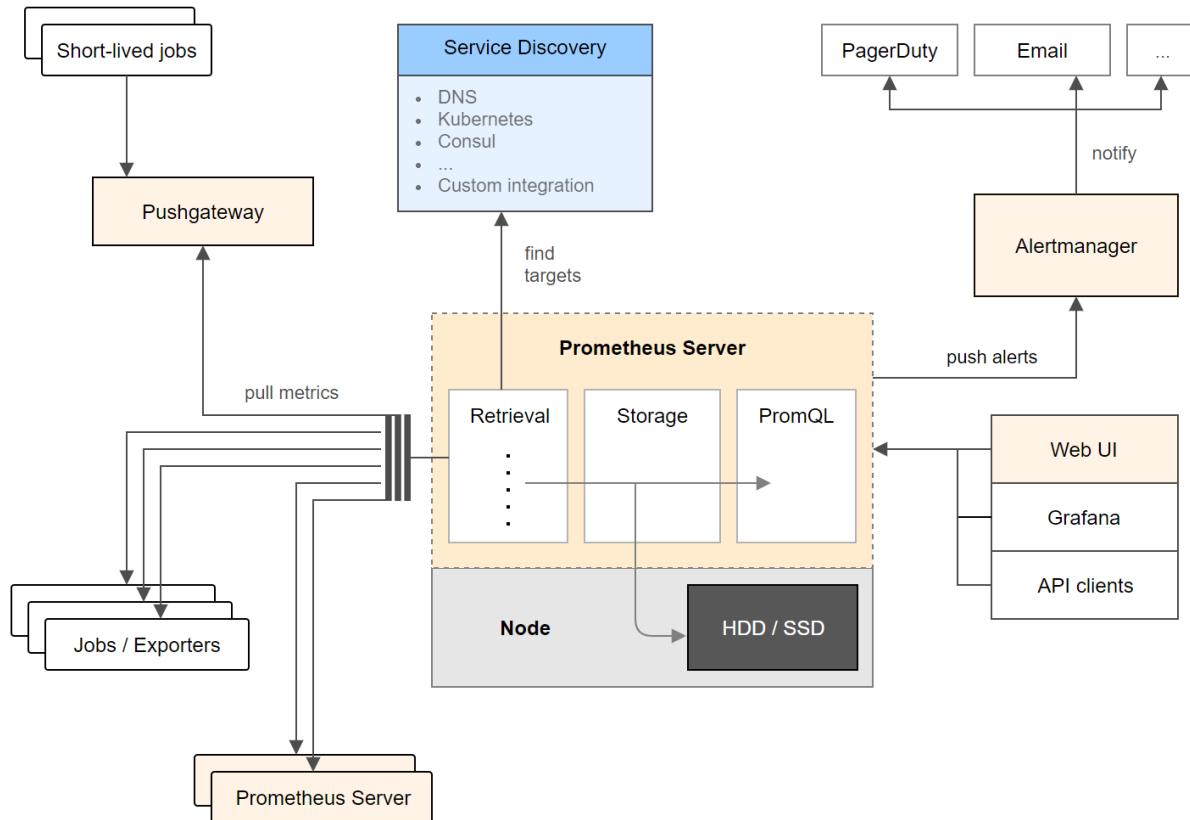


FIGURE 5.12 – Architecture de Prometheus

Lorsque la méthode *pull* ne peut pas être utilisée pour communiquer avec certains services, Prometheus propose d'utiliser le Pushgateway. Les services peuvent alors *push* les données sur le Pushgateway et Prometheus effectuera le *pull* au niveau de ce dernier. Cette solution est particulièrement intéressante pour surveiller les tablettes. En effet, le système d'exploitation Android peut mettre en pause des services afin d'économiser de la batterie (mode Doze, voir section 1.3). Cela rend la méthode *pull* inutilisable puisqu'il est impossible de prédire quand le service sera disponible. Le dispositif Android doit donc utiliser la méthode *push* pour envoyer les données.

dans les fenêtres de temps autorisés par le système d’exploitation. Malheureusement, actuellement aucun *exporter* de données n’est disponible pour Android, mais il pourrait être développé.

Prometheus peut aussi *pull* des données auprès d’une deuxième instance de Prometheus possiblement tournant dans un autre appareil.⁶ Par exemple, ceci est intéressant si des multiples instances sont déployées dans plusieurs localisations. En effet, cette fonctionnalité permet d’utiliser une instance Prometheus maître qui centralise toutes les données des instances esclaves. Un exemple de cette architecture de fédération est donné dans la figure 5.13. Cependant, une connexion à Internet est requise si les deux instances ne se trouvent pas dans le même réseau local.

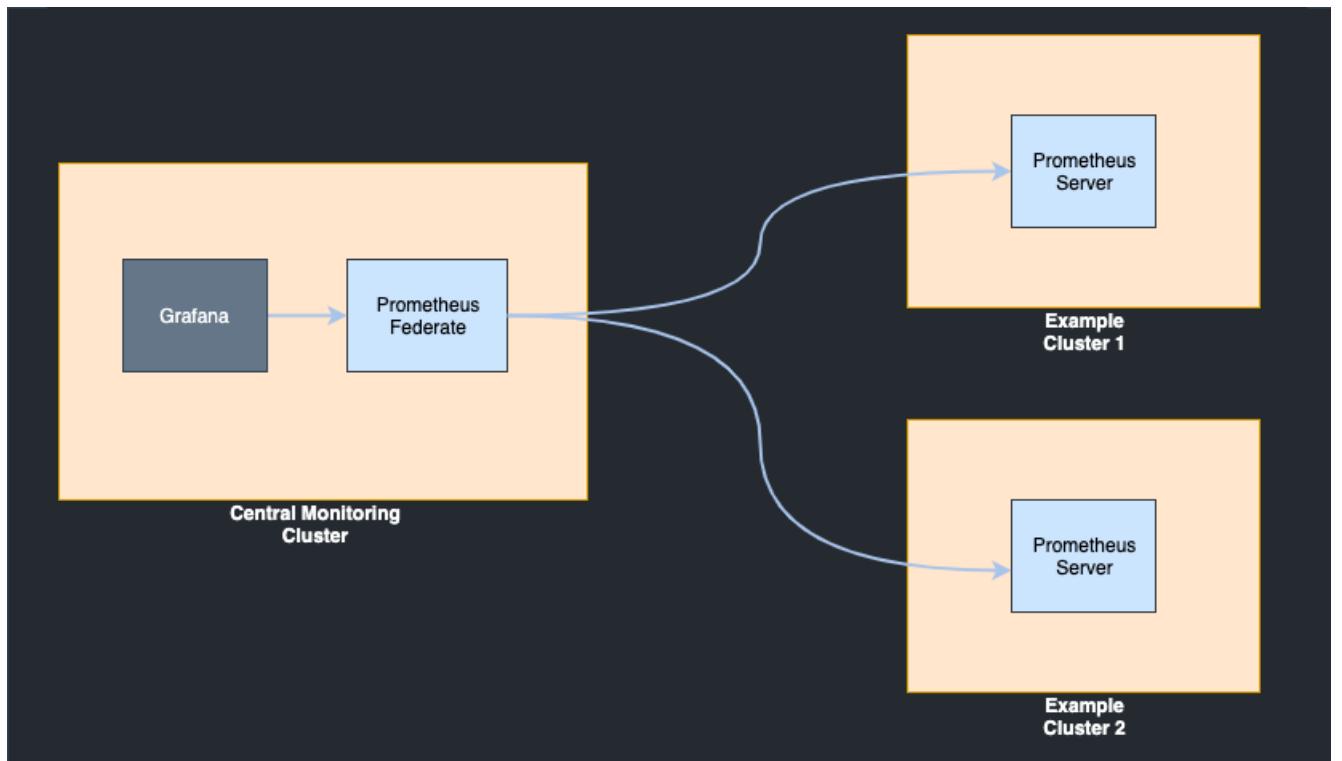


FIGURE 5.13 – Architecture de fédération de Prometheus [59]

Finalement, un des outils plus importants de Prometheus est l’Alertmanager. En effet, grâce à des fichiers de configuration YAML, il est possible de définir des conditions d’alerte et le comportement à adopter lorsqu’elles se produisent. Le Listing 1 affiche un exemple de la manière dont une telle alerte peut être configurée. En l’occurrence, cette configuration vérifie si le serveur a redémarré au cours des dernières 10 minutes et génère une alerte si c’est le cas. L’Alertmanager, comme son nom l’indique, collecte et traite toutes les alertes générées. De plus, ce dernier peut aussi transmettre des messages d’alertes par email, par messages Slack ou par des webhooks. Cette dernière fonctionnalité est très intéressante, car elle permet donc de créer un service local qui peut être contacté à travers un webhook.

C’est cette approche qui a été implémentée dans ce mémoire puisque le Raspberry Pi n’a pas accès à Internet. De ce fait, un service local, l’*alert filter and collector service*, reçoit tous les messages d’alerte envoyés par l’Alertmanager et puis il se charge de les transmettre au service de communication. Le REST API webhook de ce service a été implémenté en utilisant le web framework Flask. La figure 5.14 présente l’architecture complète de cette solution. Vous pouvez

6. support for hierarchical and horizontal federation

```

1 groups:
2   - name: Default Server Monitoring
3     rules:
4       - alert: ServerRestart
5         expr: node_boot_time_seconds != node_boot_time_seconds offset 10m
6         for: 2m
7         labels:
8           severity: critical
9         annotations:
10           summary: "Instance {{ $labels.instance }} was down very recently"
11           description: "{{ $labels.instance }} of job {{ $labels.job }} restarted
12             at least once in the past 10 minutes"
13           message: 'Serveur {{ $labels.instance }} a redémarré au moins une fois
14             lors des dernières 10 minutes'

```

Listing 1: Configuration de l'alerte relative au redémarrage du serveur

aussi remarquer que Grafana a été utilisé en tant qu'outil pour la visualisation de données. En effet, cet outil supporte la base de données de Prometheus ce qui rend le processus de configuration extrêmement simple.

Afin d'exporter les données sur l'état du serveur, les *exporters node_exporter*⁷ et *process-exporter*⁸ ont été utilisés. Ces services doivent être configurés sur la machine à surveiller. Plus de détails concernant l'installation et la configuration de ces services peut être trouvés dans [12, 13, 22]. La figure 5.16 présente un diagramme de séquence sur le processus de monitoring en utilisant le *node_exporter*. Ces deux *exporters* possèdent des dashboards déjà disponibles sur Grafana, ce qui rend la visualisation des métriques très simple. Un exemple de la visualisation des données exportées par le *node_exporter* est exposé dans la figure 5.15.

Cependant, l'utilisation de Prometheus apporte un nouveau problème au prototype. En effet, comme expliqué précédemment, la solution de communication utilise le protocole MQTT qui est basé sur le modèle publish-subscribe. En revanche, Prometheus utilise le protocole HTTP et le modèle *pull* pour échanger des données. Cela complique énormément l'envoi de données et rend impossible l'utilisation de l'architecture de fédération de Prometheus. Par conséquent, lors de ce prototype, seulement l'envoi des alertes a été implémenté. Les alertes suivent un modèle *push*, ce qui peut facilement être remplacé par un MQTT *publish*.

7. Exporte des métriques liées à l'état de la machine hôte
8. Exporte des métriques liées aux processus de la machine hôte

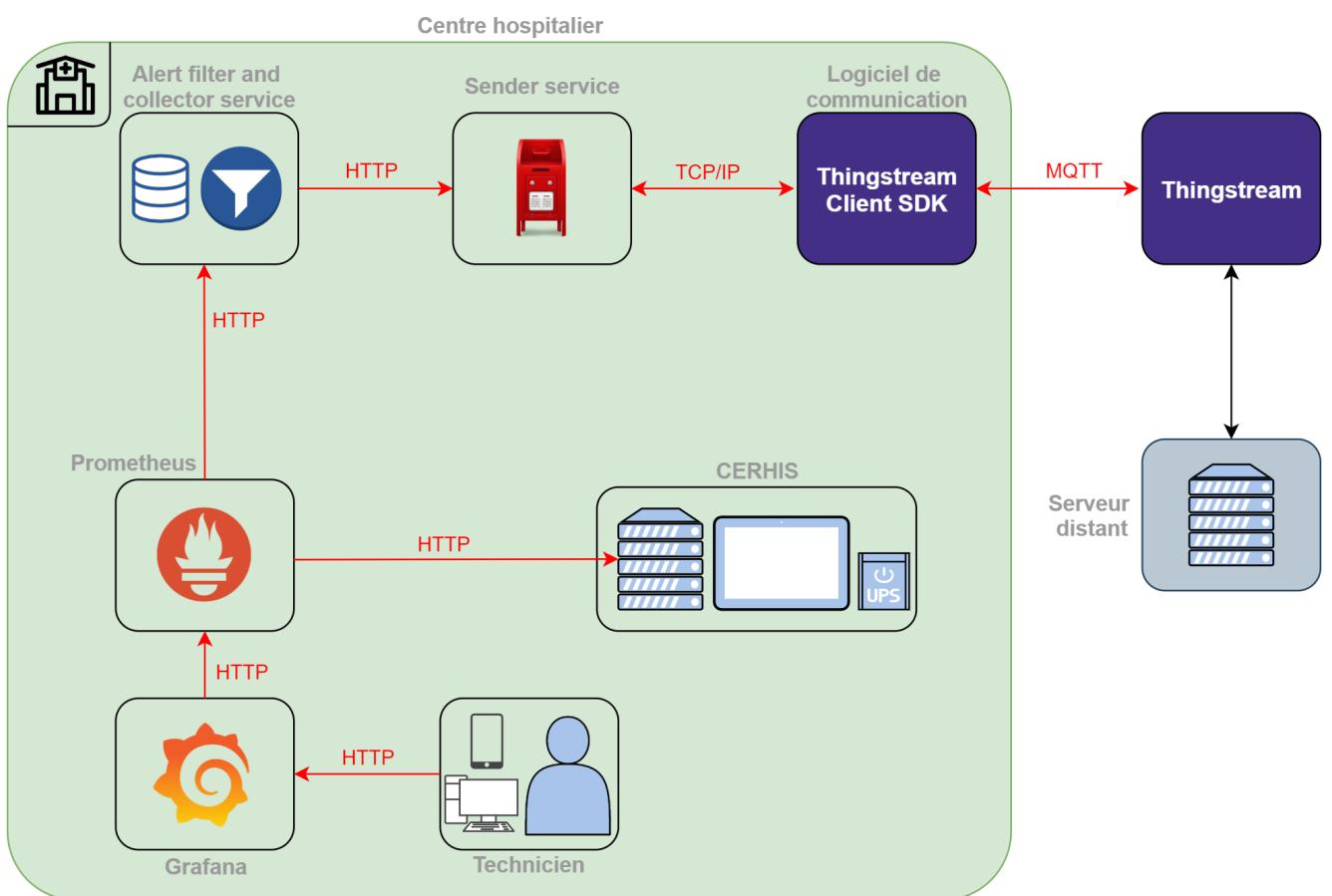


FIGURE 5.14 – Architecture du logiciel de monitoring après l'ajout de Prometheus

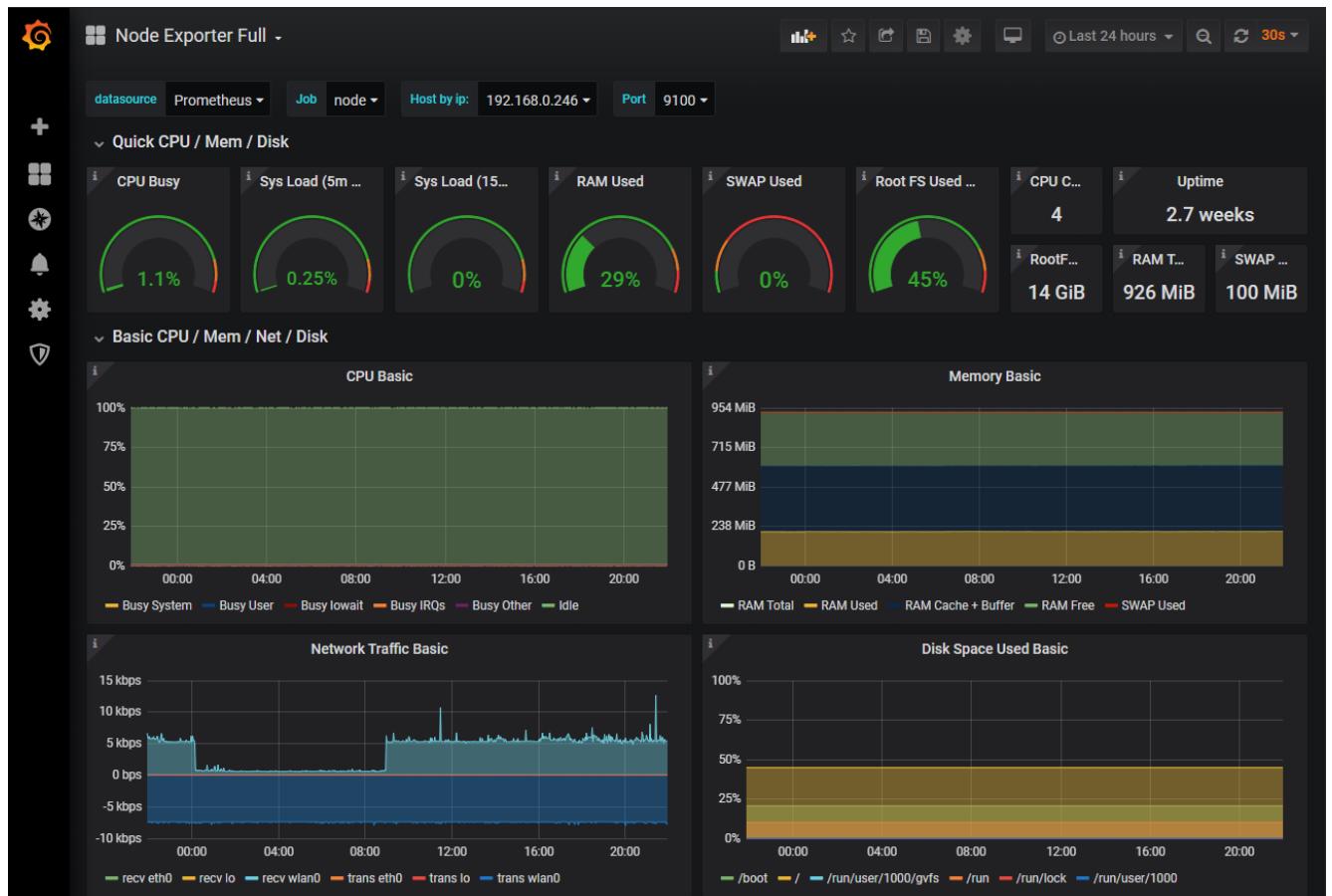


FIGURE 5.15 – Exemple de visualisation des données exportées par *node_exporter* avec Grafana

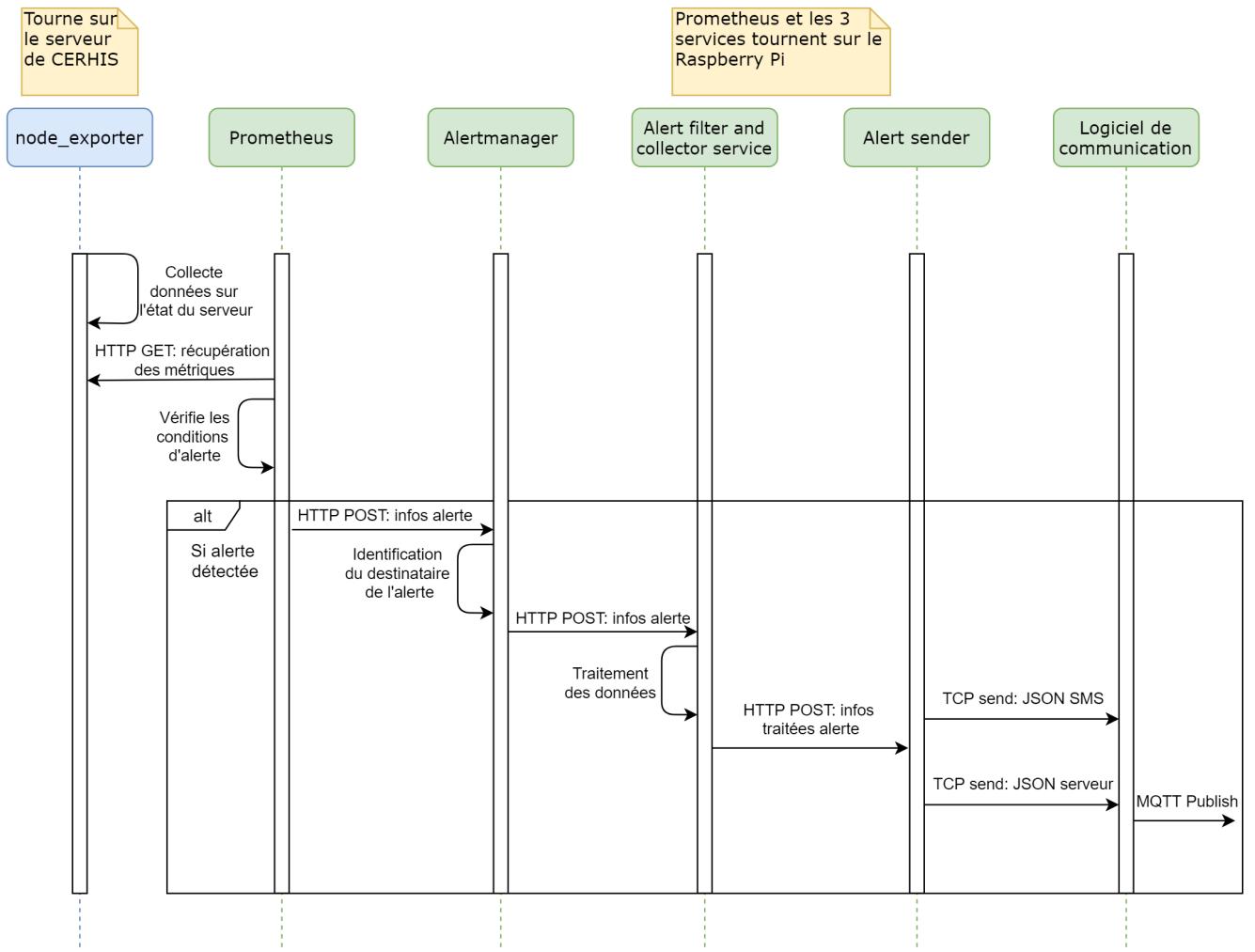


FIGURE 5.16 – Diagramme de séquence du processus de monitoring de l'état du serveur CERHIS. Ce processus est répété toutes les 15 secondes. Le MQTT publish a lieu comme décrit dans la section Envoi de données.

Mappage du réseau local

Le mappage des dispositifs connectés au réseau est une étape essentielle avant de commencer à surveiller leur présence. Cette étape pourrait être remplacée par une configuration manuelle, mais cela a plusieurs désavantages. En effet, cela demanderait qu'un technicien encode manuellement toutes les adresses IP ou MAC des dispositifs connectés. Cela peut être une possible source d'erreur qui engendrait des problèmes lors du monitoring. En outre, la configuration devrait être modifiée à chaque fois qu'une nouvelle machine est ajoutée au réseau de CERHIS.

Automatiser le processus de mappage permet donc de corriger ces problèmes. Grâce au protocole ARP⁹, mapper le réseau est très facile. Ce protocole est généralement utilisé pour traduire des adresses de la couche réseau (IP) en adresses de la couche de liaison (MAC). La figure 1 montre un exemple sur le fonctionnement de ce protocole. Brièvement, un hôte *broadcast* une requête ARP à travers le réseau local contenant une adresse IP. Si une machine de ce réseau s'est vu attribuer l'adresse IP incluse dans la requête, elle répond alors à l'hôte avec son adresse MAC. Les machines restantes reçoivent également la requête, mais comme leur adresse IP ne correspond pas, elles ignorent la demande.¹⁰

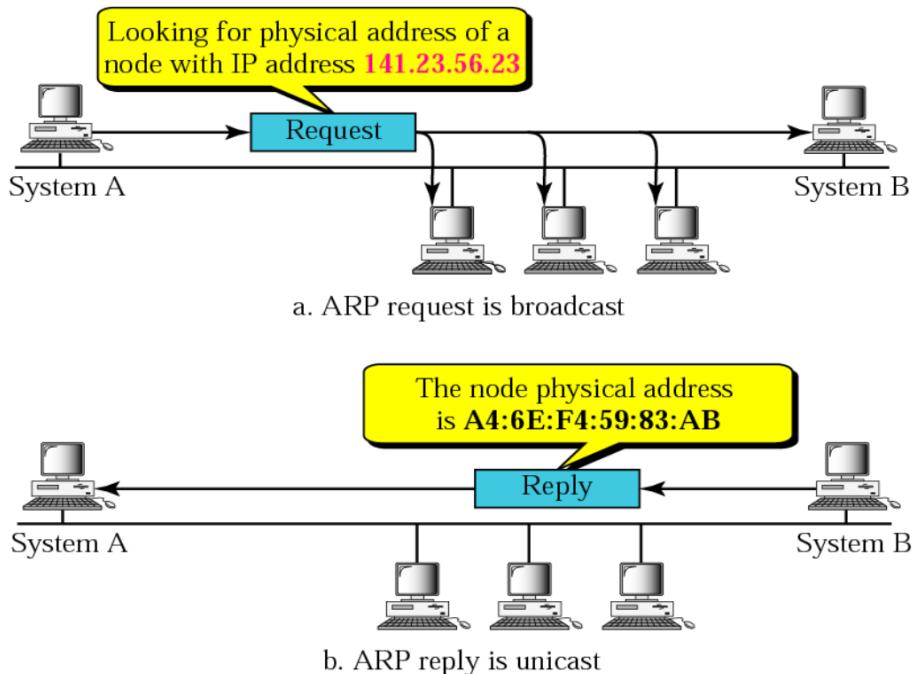


FIGURE 5.17 – Exemple de fonctionnement du protocole ARP. Dans cet exemple, le système B possède l'adresse IP 141.23.56.23 [31]

Afin de mapper le réseau, il suffit alors d'envoyer une requête ARP pour chaque adresse IP possible. Les dispositifs connectés répondront tous "présent" au moment où ils recevront une requête qui leur est destinée, permettant de réaliser le mappage assez rapidement. Le logiciel NMAP a été employé car il est open source. L'application Fing [11] a été considéré comme une alternative, car elle est très efficace pour identifier le système d'exploitant des différentes machines connectées dans le réseau. Cependant, cette application est payante, d'où la raison pour laquelle elle n'a pas été choisie.

9. Address Resolution Protocol

10. ceci présume qu'aucune personne malveillante n'est connectée dans le réseau. Pour plus d'informations à ce sujet, vous pouvez effectuer des recherches sur l'ARP poisoning

La bibliothèque logicielle de NMAP sur Python a été exploitée pour interfaçer avec le logiciel NMAP et traiter plus facilement le résultat du mappage. De plus, une base de données SQLite a été utilisée pour enregistrer la liste des dispositifs trouvés. La figure 5.18 affiche le diagramme de la table utilisé pour stocker les informations sur les dispositifs.

device		
	varchar(17)	PK
mac_address	varchar(17)	
ip	varchar(15)	
device_type	varchar(30)	
vendor	varchar(30)	NULL
discovery_date	datetime	

FIGURE 5.18 – Schéma de la table utilisée pour enregistrer les informations sur les dispositifs

Comme expliqué plus tôt, le logiciel de monitoring est constitué de plusieurs services et le mappage correspond à l'un d'entre eux. Afin de rendre le résultat accessible aux services restants, une API REST a été implémentée en utilisant le web framework Flask. Ceci permet aux services qui nécessitent de la liste de dispositif de l'obtenir à partir d'une requête HTTP GET.

Finalement, lorsque le service de mappage a été testé pour la première fois, il n'était pas capable d'identifier tous les dispositifs connectés sur le réseau. En particulier, cela semblait toujours se produire avec le même sous-ensemble d'appareils Android qui était connecté au réseau à travers le Wi-Fi. Ceci était un énorme problème puisque CERHIS possède des tablettes Android. Pour tenter de résoudre ce souci, la première proposition fut de réduire la périodicité de l'envoi des requêtes ARP. Ceci a permis de détecter quelques dispositifs supplémentaires, mais quelques smartphones continuaient à poser le même problème. La deuxième technique a été de suivre le chemin inverse. En effet, au lieu de diminuer la cadence, l'idée fut de réaliser plusieurs mappages consécutifs pour inonder le réseau avec plusieurs requêtes. Grâce à cette deuxième approche, tous les appareils ont pu être trouvés. Finalement, la solution implémentée est un hybride de ces deux techniques. Premièrement, un scan avec une très faible cadence est exécuté. Ensuite, trois mappages sont effectués en succession. Lors des tests réalisés, cette dernière solution s'est montrée 100% efficace.

Finalement, la décision a été prise de ne scanner le réseau qu'une seule fois lorsque le service est lancé. On peut s'attendre à ce que l'ensemble des appareils de CERHIS reste assez constant dans le temps. En outre, scanner le réseau en continu conduirait à deux situations difficiles à gérer qui sont :

- En cas de disparition d'un appareil, il ne serait pas trouvé durant le scan plus récent. Quelle action devrait alors être adoptée dans cette situation ? Retirer le dispositif de la liste ou ignorer sa disparition ? Il est impossible de savoir exactement ce qui s'est passé et adapter un des deux comportements ne permettrait pas de résoudre entièrement le problème.
- Réaliser un balayage en continu rendrait la connexion temporaire d'appareils très difficile. En effet, un technicien pourra nécessiter de connecter son ordinateur ou smartphone pour accéder au serveur ou à l'interface de visualisation de données. Si un scan en continu est effectué, ce dispositif serait détecté et ajouté à liste d'appareils de CERHIS, même s'il n'en fait pas partie.

- Enfin, un balayage du réseau utilise une grande quantité de ressources sur la machine qui exécute le service, mais aussi sur les dispositifs du réseau puisque les trames (frames) sont broadcast.

Si un appareil doit être ajouté ou retiré du réseau, il suffira alors de redémarrer manuellement le service et les modifications seront prises en compte assez rapidement.

Surveiller la présence des divers composants

Une fois le mappage du réseau terminé, la liste des machines connectées peut être utilisée pour connaître les adresses IP et MAC des dispositifs à surveiller. Généralement, le protocole ICMP ECHO, aussi connu sous le nom de Ping, est employé afin de savoir si un dispositif est toujours connecté au réseau. Des requêtes d'ECHO (*echo-request*) sont transmises par le dispositif désirant connaître s'il sait contacter l'appareil destinataire. Ensuite, le destinataire reçoit la demande et envoie une réponse echo (*echo-reply*) à l'émetteur. Ce procédé est résumé sur la figure 5.19.

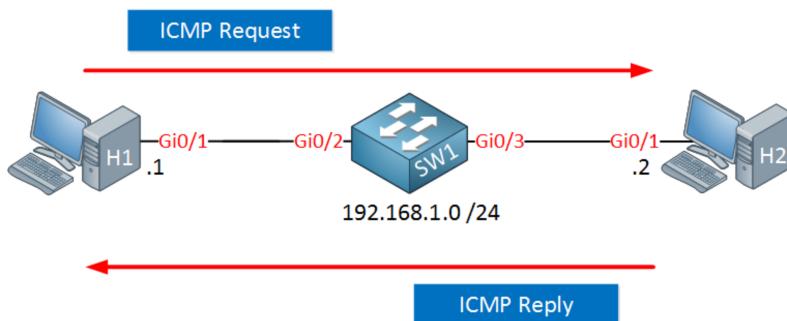


FIGURE 5.19 – Exemple d'un Ping (ICMP *echo-request* et ICMP *echo-reply*)

Cependant, de nombreux pare-feux bloquent ce type de trafic, ce qui rend impossible de connaître si l'appareil est connecté au réseau. En revanche, le protocole ARP n'est presque jamais bloqué, car il est nécessaire afin que les dispositifs puissent communiquer. Mais, comme mentionné ci-dessus, les demandes ARP sont broadcast à travers tout le réseau, ce qui peut générer beaucoup de trafic et avoir un impact sur les performances de ce dernier. De ce fait, dans un premier temps, NMAP a été employé pour monitorer les appareils. Cela permettrait de laisser le logiciel choisir la méthode idéale pour surveiller chaque dispositif.

Cette implémentation s'est avérée très efficace à détecter si un appareil était connecté. Au contraire, elle ne s'est pas montrée aussi fonctionnelle du point de vue de l'utilisation des ressources, créant des pics d'utilisation du CPU de l'ordre de 60%. Sachant que les pings doivent être effectués assez fréquemment, cette méthode a dû être abandonnée. En effet, une utilisation élevée du processeur aurait entraîné une haute consommation d'énergie, ce qui aurait engendré des problèmes lorsque l'appareil serait alimenté par une batterie.

La bibliothèque logicielle `scapy` a été manipulée par la suite pour concevoir une solution plus adaptée au contexte de ce projet. `Scapy` permet de générer manuellement des paquets et des trames de plusieurs protocoles. Puisque les adresses IP et MAC sont connues (grâce au scan), il est alors très facile de créer un paquet ICMP et de l'encapsuler par une trame qui contient l'adresse MAC du destinataire. Ceci retire la responsabilité au système d'exploitation de créer la trame et optimise tout le processus. En effet, si jamais la MAC ne se trouve pas dans la table ARP, le système d'exploitation doit broadcast une requête ARP. Ajouter manuellement l'adresse MAC évite ce broadcast.

Cette méthode fonctionne avec la majorité des dispositifs, mais ne résout pas le problème lorsque le pare-feu de l'appareil filtre les paquets ICMP. De ce fait, si un dispositif ne répond pas à une demande d'ECHO après 2 secondes, une requête ARP contenant l'adresse IP de ce dispositif est broadcast dans le réseau. La combinaison de ces deux procédés permet d'avoir un taux de succès très élevé. Cependant, même en recourant à ces deux techniques, il est arrivé que certains smartphones Android ne répondissent à aucune des demandes. Ce comportement était assez rare, mais avait tout de même un impact sur la qualité de la solution. Pour le résoudre, une approche assez similaire à celle du mappage du réseau a été utilisée. Lorsque l'appareil ne réagit à aucune des deux requêtes précédentes, de multiples paquets ICMP sont envoyés en succession pendant une période de 4 secondes. Ces paquets sont également encapsulés dans une trame qui contient l'adresse MAC du destinataire. Enfin, la combinaison de ces trois techniques a permis d'atteindre une très haute bonne détection de la présence des dispositifs. De plus, il n'a pas été possible de reproduire le même problème avec les smartphones Android lorsque les 3 techniques sont utilisées. Un diagramme de flux du processus de monitoring est présenté dans la figure 5.20.

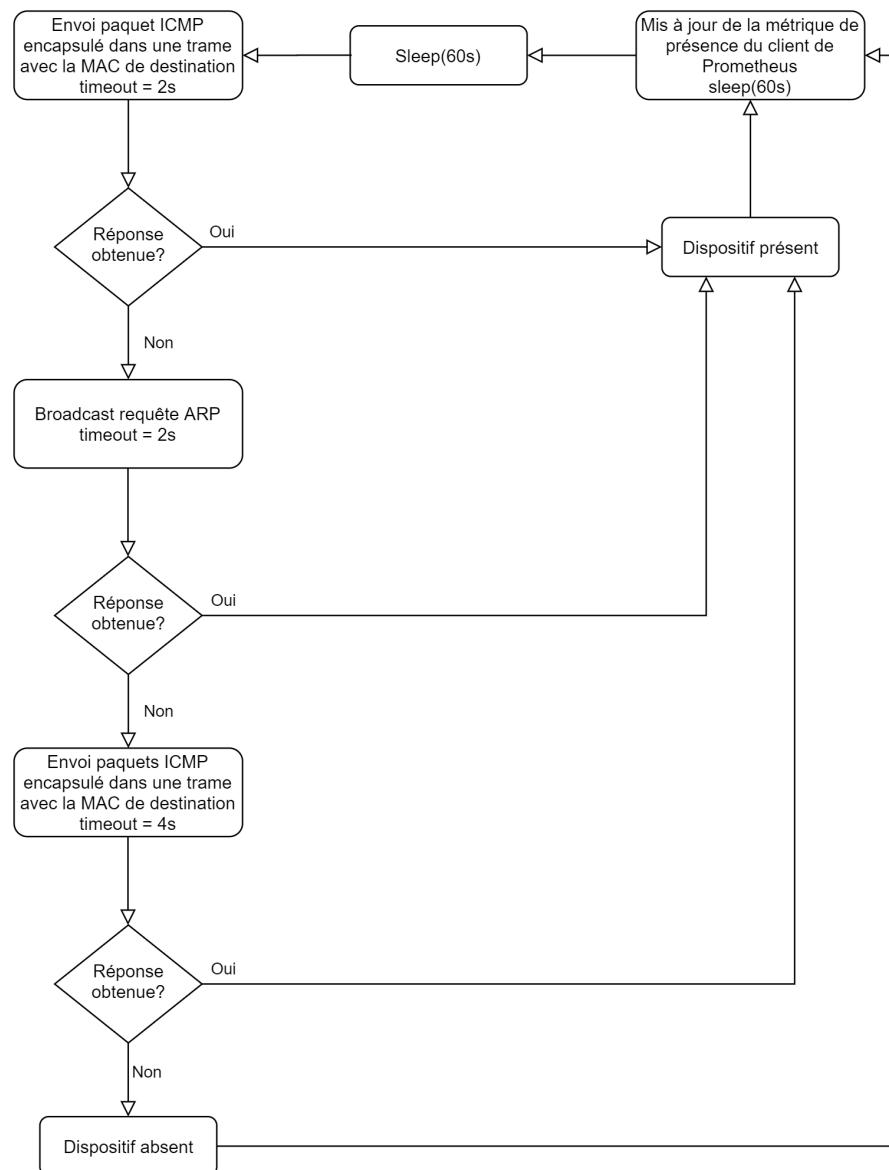


FIGURE 5.20 – Diagramme de flux du processus de vérification de la présence d'un dispositif

Finalement, il fallait implémenter la sauvegarde des données de présence et produire les alertes en cas de disparition d'un appareil. Cela aurait pu être développé manuellement. Mais, puisque Prometheus était déjà déployé comme base de données et gestionnaire d'alertes, l'utiliser également pour stocker les informations sur la présence des appareils semblait être la solution plus naturelle. En outre, cela permet aussi d'exploiter toutes les capacités de gestion d'alertes pour configurer et générer des messages lorsqu'un dispositif est manquant. Pour exporter les données de présence, le client Prometheus pour Python a été utilisé. L'architecture finale de la solution de surveillance est exposée dans la figure 5.21.

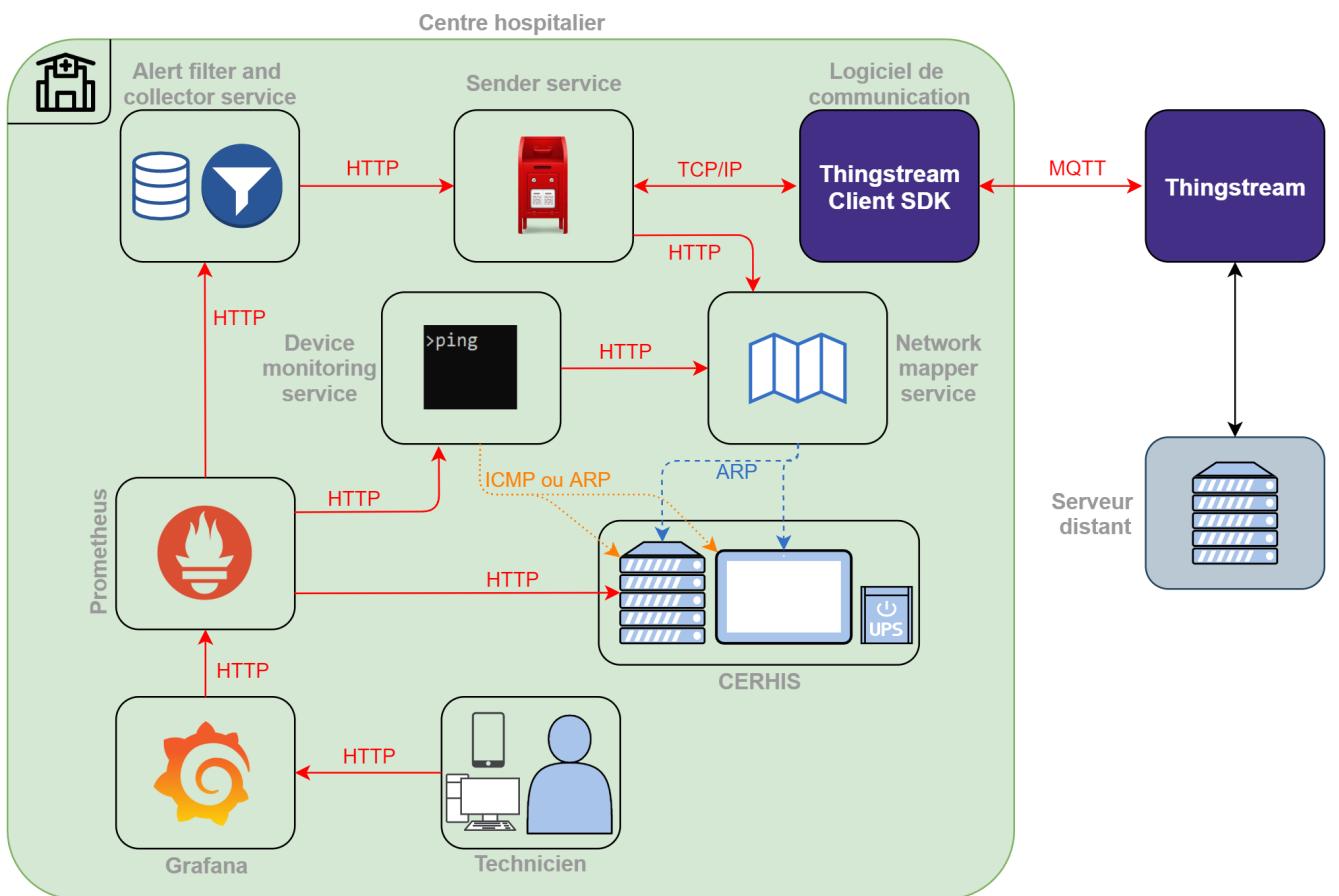


FIGURE 5.21 – Architecture complète du client de monitoring

Maintenant que l'entièreté de la solution de monitoring a été présentée, il est possible de revenir sur quelques détails qui ont été omis sur les sections antérieures pour faciliter les explications. Premièrement, la configuration actuelle de Prometheus permet d'identifier et envoyer des messages pour quatre types d'alertes qui sont :

- **ServerRestart** : Cette alerte emploie les données exportées par le *node_exporter* afin de repérer si le serveur a redémarré au cours des 10 dernières minutes.
- **Process Down** : Détecte si un des processus surveillés par le service *process-exporter* a été interrompu. La liste des process monitorés est définie dans le fichier de configuration de *process-exporter*.
- **DeviceDown** : Exploite les données relatives à la présence des dispositifs exportées par le service device monitoring pour vérifier si un appareil est absent depuis plus de 2 minutes.
- **InstanceDown** : Cette alerte a lieu si un des services *exporters* utilisés par Prometheus pour obtenir les données cesse de répondre aux requêtes HTTP.

Ces quatre alertes correspondent à ce qui était initialement demandé dans le cahier des charges. À l'avenir, d'autres alertes pourront être ajoutées par le biais des fichiers de configuration de Prometheus, comme celui illustré dans le listing 1.

Deuxièmement, le *sender service* ne se limite pas à transférer les alertes envoyées par le logiciel *alert filter and collector*. En effet, lors de son initialisation, le *sender service* essaie de contacter le *network mapper* afin d'obtenir la liste de dispositifs. Cette liste est ensuite transmise au service de communication qui l'envoie au serveur.

Enfin, l'*alert filter and collector service* emploie une base de données SQLite afin de sauvegarder de façon permanente toutes les alertes générées par Prometheus. En cas de défaillance de la solution de communication, les techniciens pourront récupérer le fichier de la base de données et avoir accès à un historique des événements, même plusieurs jours après qu'ils se soient produits. Le schéma de la base de données est illustré dans la figure 5.22.

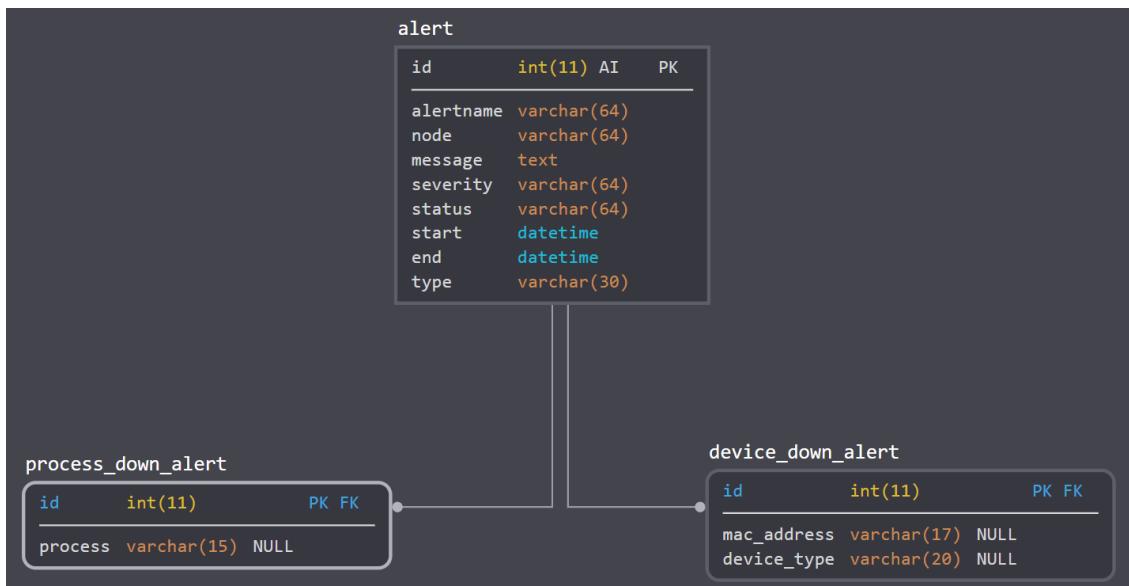


FIGURE 5.22 – Schéma de la base de données du *alert filter and collector service*

5.1.3 Sécurité

La sécurité ne figurait pas parmi les objectifs du cahier des charges, d'où la raison pour laquelle elle n'a pas vraiment été prise en compte lors de la réalisation de ce prototype. En fait, les hypothèses suivantes ont été faites :

- Aucun agent malveillant n'est connecté au réseau local de CERHIS.
- Aucun dispositif malveillant ne peut se connecter au réseau de Thingstream. En particulier, aucun agent malveillant ne pourra se faire passer par un client SN-Thing.

Cette dernière hypothèse se traduit donc par une confiance totale dans le système d'authentification de la carte SIM de Thingstream et du réseau GSM. Cependant, dans une version finale du client, faire cette hypothèse ne serait peut-être pas judicieux. Il est fortement recommandé qu'une technique d'échange de clés soit implémentée entre le serveur et le client de communication afin que les transmissions entre les deux instances puissent être cryptées. Ceci empêcherait donc d'être complètement du système de sécurité de Thingstream qui nous est inconnu. En outre, le chiffrement des données permettrait d'être protégé contre des attaques sur le réseau GSM. Ces attaques ne sont pas faciles à conduire, mais peuvent potentiellement exposer les informations transmises dans le réseau. [41]

5.2 Serveur

Le serveur distant est une partie intégrale de la solution finale. Il permet aux techniciens de consulter l'état actuel de l'infrastructure même s'ils ne se trouvent pas dans le centre hospitalier. Les SMS ne sont utilisés que pour les prévenir en cas de défaillance d'un des composants de CERHIS. Cependant, certaines de ces défaillances peuvent être de courte durée, voire même un faux positif. En outre, les SMS n'offrent pas une vue d'ensemble sur la liste des composants et l'historique des alertes. Le serveur cherche donc à surmonter ces problèmes en sauvegardant toutes les informations sur une longue période et en proposant un accès facile aux données à travers une interface graphique. Les informations sur le serveur seront également mises à jour en cas de changement, mais cela ne se fera pas par SMS.

L'application s'exécutant sur le serveur est beaucoup plus simple que celle du client. En effet, le serveur ne fait que : recevoir des informations déjà traitées par ce dernier, les stocker sur la base de données et les rendre accessibles aux techniciens. De ce fait, l'architecture du logiciel est composée par les trois parties suivantes :

- Le client MQTT, ou Thingstream IP-Thing, qui reçoit tous les messages publiés sur les topics **device/#identity#/startup** et **device/#identity#/post**.
- Une base de données capable de stocker toutes les informations envoyées par plusieurs clients.
- Une application ou un service permettant de facilement visualiser les données se trouvant sur la base de données.

La figure 5.23 illustre cette architecture. L'implémentation de chaque composant sera exposée au cours des trois sections suivantes.

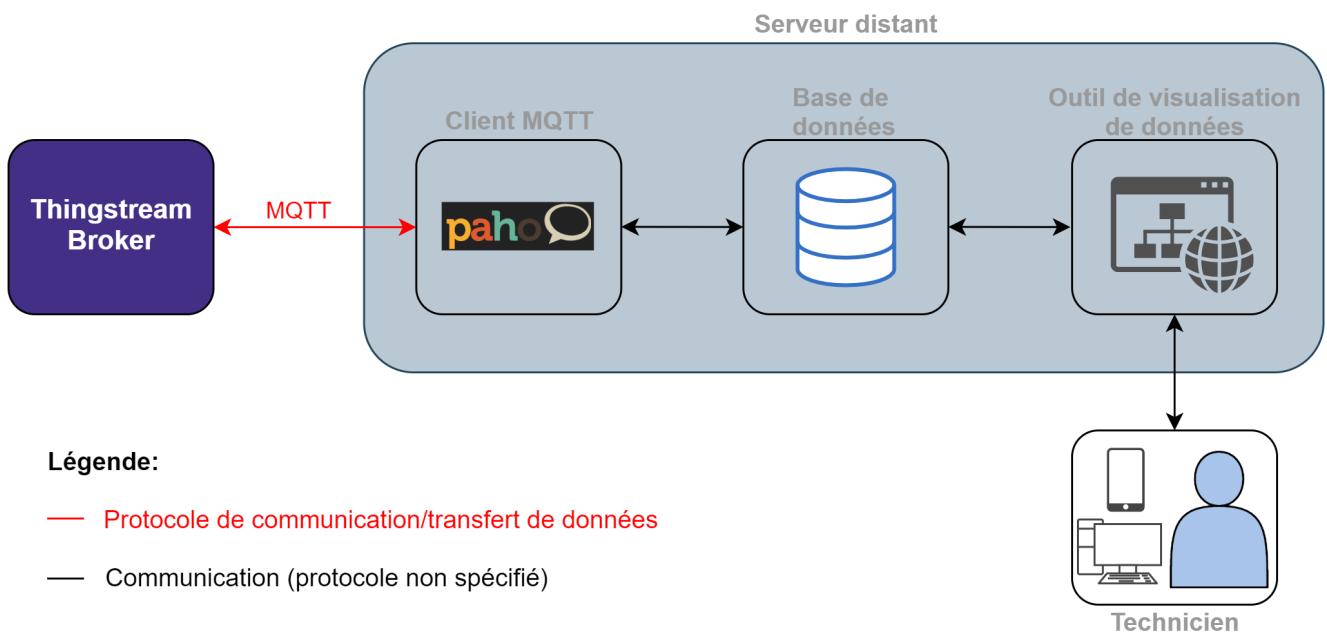


FIGURE 5.23 – Architecture simplifiée de l'application tournant sur le serveur distant

5.2.1 Base de données

De nos jours, plusieurs types de bases de données existent sur le marché, mais elles sont très souvent classées selon deux modèles : le modèle relationnel et les bases de données NoSQL (non relationnelles). Ces dernières abandonnent les propriétés ACID¹¹ en faveur d'une plus haute disponibilité, de bonnes performances même lorsqu'une grande quantité de données doit être traitée, et l'extensibilité¹².

Cependant, en raison des limitations de la solution de communication, seulement des mises à jour concernant l'état des alertes et la liste des dispositifs seront envoyées au serveur. Les métriques collectées par Prometheus ne seront stockées que sur le Raspberry Pi. Cet aspect a une grande influence sur le choix de la base de données. En effet, ceci diminue très fortement la quantité de données que le serveur devra enregistrer. En outre, toutes les informations envoyées par le client sont également stockées sur celui-ci sur des bases de données SQLite. Cela veut donc dire que les données transmises sont déjà formatées selon le modèle relationnel. Il apparaît tout à fait évident que la base de données du serveur suive le même modèle, puisque cela réduit le nombre de technologies à maîtriser pour travailler sur ce projet.

Le schéma de la base de données du serveur peut donc être très similaire à ceux présentés sur les figures 5.18 et 5.22. La grande différence c'est que les diverses tables peuvent être combinées sous un même schéma. Une base de données SQLite pourrait également être utilisée du côté du serveur, mais cela présenterait énormément de problèmes plus tard lorsque celui-ci recevrait des informations de multiples clients. SQLite a une très faible extensibilité et est optimisé pour les petites bases de données. Cependant, la base de données du serveur doit pouvoir stocker les informations de plusieurs clients. Le choix de SQLite n'est donc pas le plus judicieux pour ce projet.

11. atomicité, cohérence, isolation et durabilité

12. *scalability* en anglais

Il existe sur le marché des alternatives qui sont également open source et plus puissantes que SQLite. Contrairement à cette dernière, ces bases de données ne sont pas autonomes, elles ont besoin d'un serveur¹³ pour fonctionner. Elles suivent donc le modèle client-serveur pour échanger des données. Les 3 principales alternatives sont :

- MySQL
- PostgreSQL
- MariaDB

N'importe laquelle des trois possibilités aurait pu être déployée. Ce projet ne possède pas beaucoup de contraintes sur cet aspect et la structure des données à stocker n'est pas complexe. MySQL a été choisi parce que c'est une des plus répandues et, traditionnellement, celle qui est employée quand quelqu'un désire apprendre à manipuler des bases de données. En outre, puisque son utilisation est très habituelle, une grande majorité des outils de visualisation sont compatibles avec ce choix. Ce dernier aspect sera important plus tard lors de la définition de l'outil de visualisation de données.

Afin de communiquer avec la base de données, la bibliothèque logicielle SQLAlchemy a été manipulée. Elle permet de faire abstraction de toute la partie liée aux requêtes SQL. Cela a contribué à réduire le temps de développement et à diminuer la complexité du code. Le schéma de la base de données du serveur distant est illustré dans 5.24. Veuillez noter l'ajout du *thingstream_id* dans chaque table, ce qui permet de différencier les données en fonction du centre hospitalier. En outre, la table *thingstream_device* a également été additionnée pour enregistrer les valeurs liées à la vérification de l'identité d'une application.

5.2.2 Implémentation des fonctionnalités

Client MQTT

Tout comme la base de données, le logiciel du serveur est assez simple. La bibliothèque logiciel Python Eclipse Paho MQTT a été utilisé pour implémenter le client qui reçoit tous les messages publiés par les SN-Things. Cette bibliothèque possède énormément de documentation et d'exemples d'utilisation en ligne.

Comme expliqué au début de cette section, ce client est abonné à deux topics. Le premier, **device/#identity#/startup**, est le topic utilisé par les SN-Things lorsqu'une application essaye de s'authentifier auprès du serveur. Afin de vérifier l'identité, le serveur compare l'id envoyé par l'application et la valeur **#identity#** avec des valeurs encodées au préalable sur la base de données. En cas de correspondance (non-correspondance), le serveur répond avec un message autorisant (refusant) la demande de connexion de l'application. Ce message est lu par le client de communication tournant sur le Raspberry Pi qui, en fonction de la réponse, décidera ensuite de maintenir ou fermer la connexion avec l'application .

Le deuxième, **device/#identity#/post**, correspond au topic employé pour transmettre les données relatives aux alertes et aux dispositifs. Dans ce cas, le serveur lit d'abord le type des données et ensuite enregistre les sur la table de la base de données qui correspondant au type présent dans l'en-tête.

13. Ce serveur peut correspondre à la même machine du serveur distant ou à une seconde machine complètement différente. Jusqu'à présent, le mot serveur a été utilisé pour désigner l'ensemble constitué par le client MQTT, la base de données et la visualisation de données. En revanche, lors de la communication entre ces composants, certains jouent le rôle de client et d'autres celui du serveur. Par exemple, la solution de visualisation de données est un client qui se connecte au serveur de la base de données. Désormais, si les mots serveur et client sont soulignés, alors ils correspondent à ce dernier cas et non pas aux solutions créées dans ce mémoire.

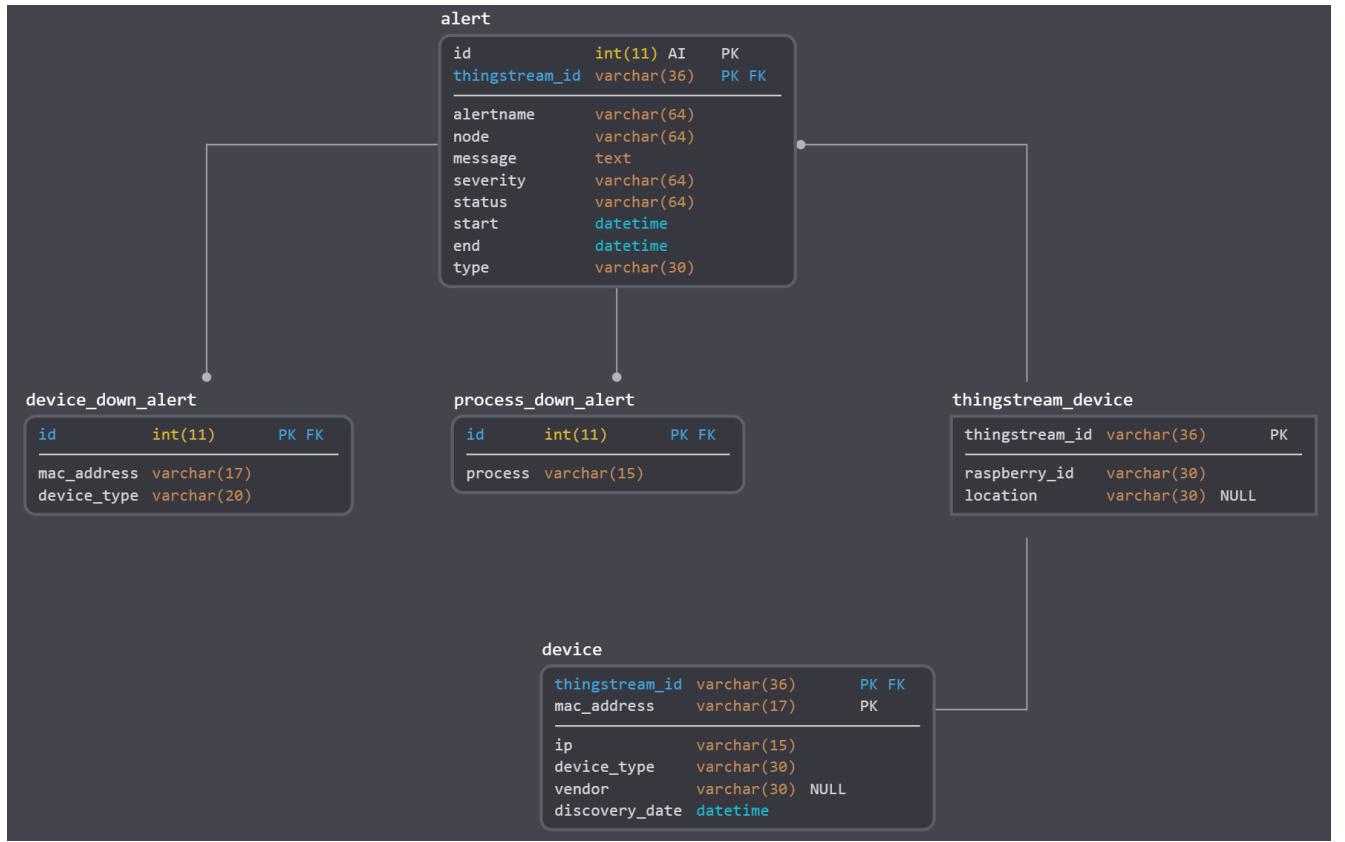


FIGURE 5.24 – Schéma de la base de données du serveur

Afin de s'adresser à un SN-Thing en particulier, le serveur publie des messages sur le topic **device/#identity#/recv**. Actuellement, seulement le message d'acceptation ou de refus de la connexion est transmis. Ce message prend la forme suivante :

```

1 {
2   "id" : #id#,
3   "accepted" : #valeur booleenne#
4 }
```

Finalement, la figure 5.25 illustre le processus lié à la vérification de l'identité d'une application.

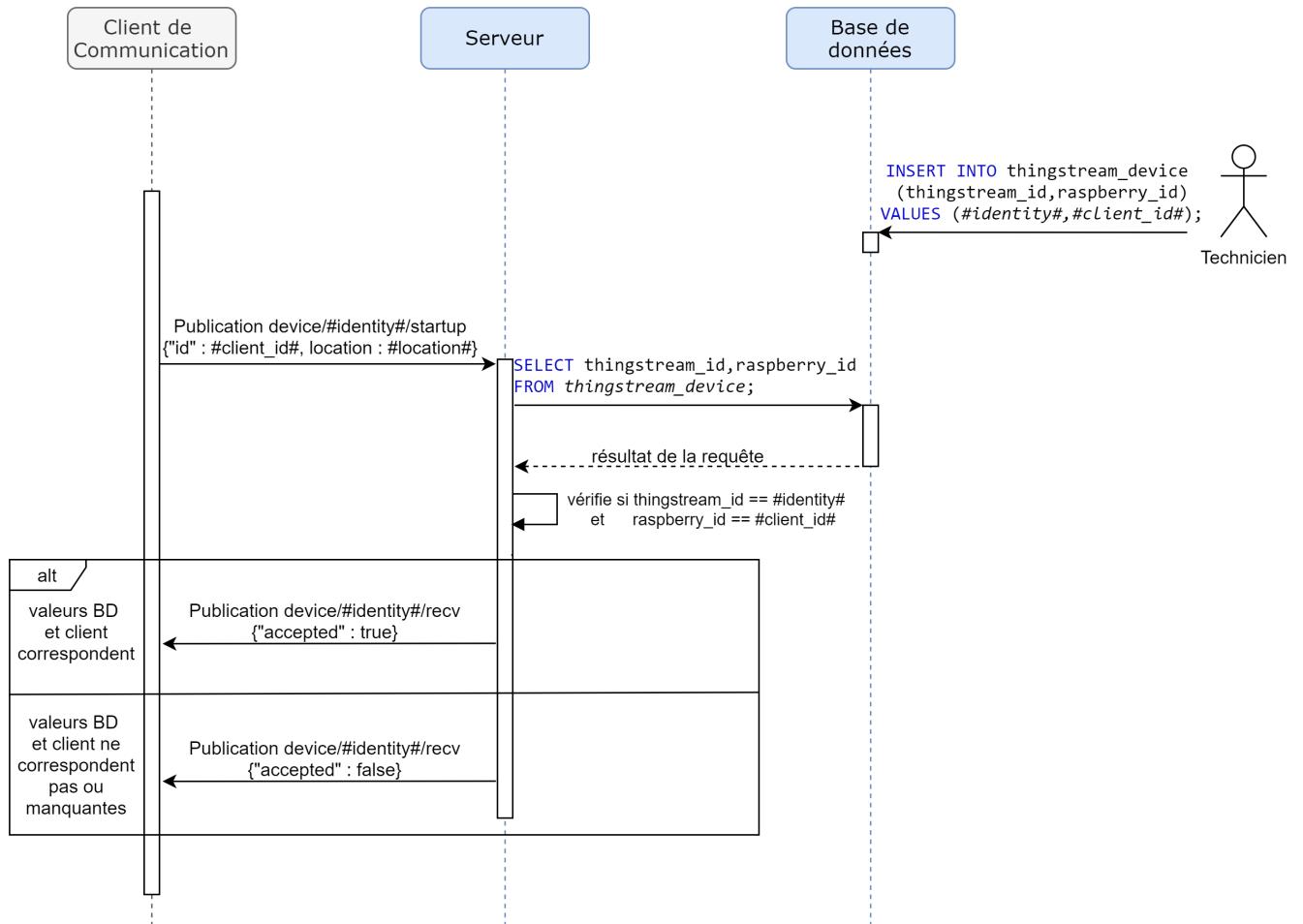


FIGURE 5.25 – Processus de vérification de l'identité d'une application par le serveur. Ce diagramme complémente celui de la figure 5.11

Interface graphique

Le dernier point lié à l'application du serveur correspond à l'outil de visualisation de données. Il est primordial que les techniciens puissent vérifier l'état de l'infrastructure de CERHIS et trouver la source de la défaillance, s'il y en a une. En conséquence, un outil adapté aux données du projet est requis. Ils peuvent être classés sous trois grandes catégories : les bibliothèques logicielles, les dashboard, et les outils Business Intelligence (voir section 2.3).

Les outils Business Intelligence ont été abandonnés en premier. Généralement, leur utilisation est associée avec un coût élevé et la quantité de données traitée actuellement ne requiert pas l'emploi d'un tel outil. En outre, le cahier des charges ne spécifie pas la nécessité de trouver une tendance dans les données.

La décision se portait donc sur l'emploi d'un d'un outil *dashboard* ou d'une bibliothèque logicielle. Cette dernière permet de créer une solution complètement personnalisée pour le projet, tirant le meilleur parti des données. En revanche, partir sur cette route demande une charge de travail importante. Il y a toute une logistique associée à cette approche comme le développement d'un site web, ou le choix de la meilleure manière pour afficher les données. En particulier, pour ce dernier point, il faut prendre en compte les 3 principes d'un bon outil de visualisation de données. (Fiables, Accessibles, et Élégantes).

Ces mêmes principes doivent être suivis lors de l'emploi d'un outil *dashboard*. Mais, le grand avantage est que ces outils aident l'utilisateur à accomplir cet objectif en offrant déjà certaines visualisations prédéfinies. De plus, ces affichages peuvent s'ajuster automatiquement aux données (par exemple les axes d'un graphique). Finalement, les *dashboard* possèdent aussi un certain degré de customisation afin de laisser l'utilisateur adapter la visualisation à ses besoins. Tout cela est souvent proposé sous la forme d'un logiciel facile à installer et ne demandant pas de connaissances en programmation.¹⁴ Le problème majeur réside dans le fait que ces outils peuvent parfois ne pas offrir un certain affichage qui est recherché.

Enfin, Grafana a été choisi comme solution de visualisation. Cet outil *dashboard* s'est montré simple à manipuler lors de sa configuration pour la partie du client et l'utiliser pour le serveur permet d'avoir un ensemble de solutions plus homogène. De plus, il est compatible avec la base de données MySQL choisie antérieurement. À ce stade du prototype, créer une solution personnalisée n'est pas le choix le plus pertinent. L'application du serveur risque d'évoluer énormément durant les prochaines étapes si de nouvelles révisions sont apportées au client. Partir d'un *dashboard* offre une immense flexibilité, car la visualisation peut être modifiée en une question de minutes, contrairement à l'utilisation d'une bibliothèque logicielle.

À l'avenir, Grafana pourrait être remplacé si ce choix s'avère être un facteur limitant du projet, mais étant donné le grand nombre de plugins disponibles et qui ne fait qu'augmenter, cela pourrait ne jamais arriver.

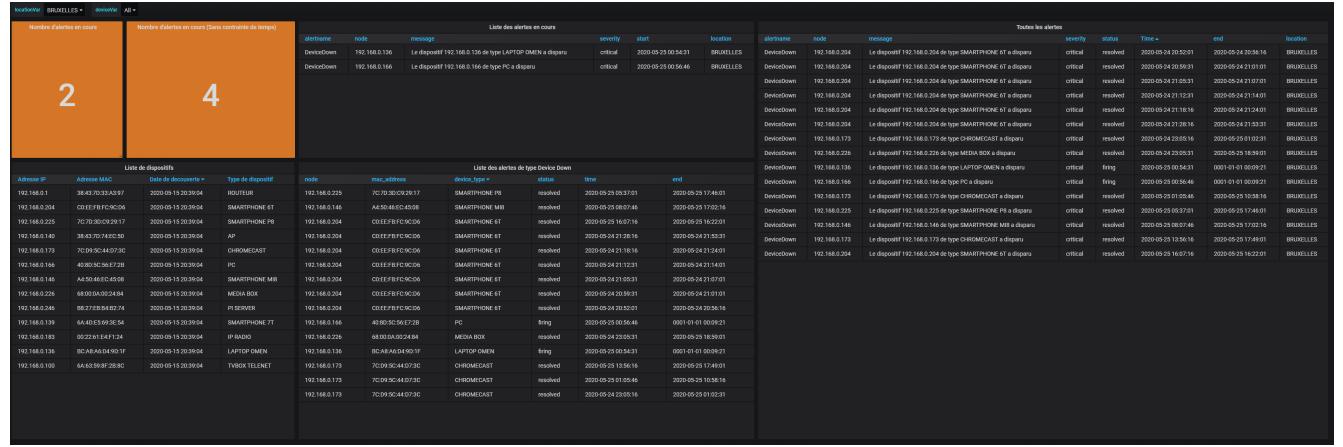


FIGURE 5.26 – Interface de visualisation de données du serveur distant

5.2.3 Sécurité

Contrairement au client, le serveur a un accès à Internet. Le chiffrement des données devient beaucoup plus important afin de protéger les échanges d'informations. En particulier, les identifiants requis pour se connecter à Grafana ne doivent en aucun cas être transmis sur Internet sans utiliser des méthodes de cryptographie. De ce fait, le protocole TLS a été employé pour sécuriser la communication entre tous les composants du logiciel. La figure 5.27 montre le fonctionnement de ce protocole. Une fois qu'une connexion est établie entre deux parties (client-serveur), les deux instances échangent des clés de chiffrement qui seront utilisées pour crypter les données. En outre, le serveur envoie également un certificat qui permet de vérifier son identité auprès d'une

14. La manipulation des langages de requête (SQL) peut être nécessaire si la requête souhaitée ne peut être construite à l'aide de constructeurs de requêtes (*query builders*).

tierce partie de confiance (l'instance qui a délivré le certificat). Cette vérification évite que des attaques de l'homme du milieu (voir figure 5.28) puissent avoir lieu.

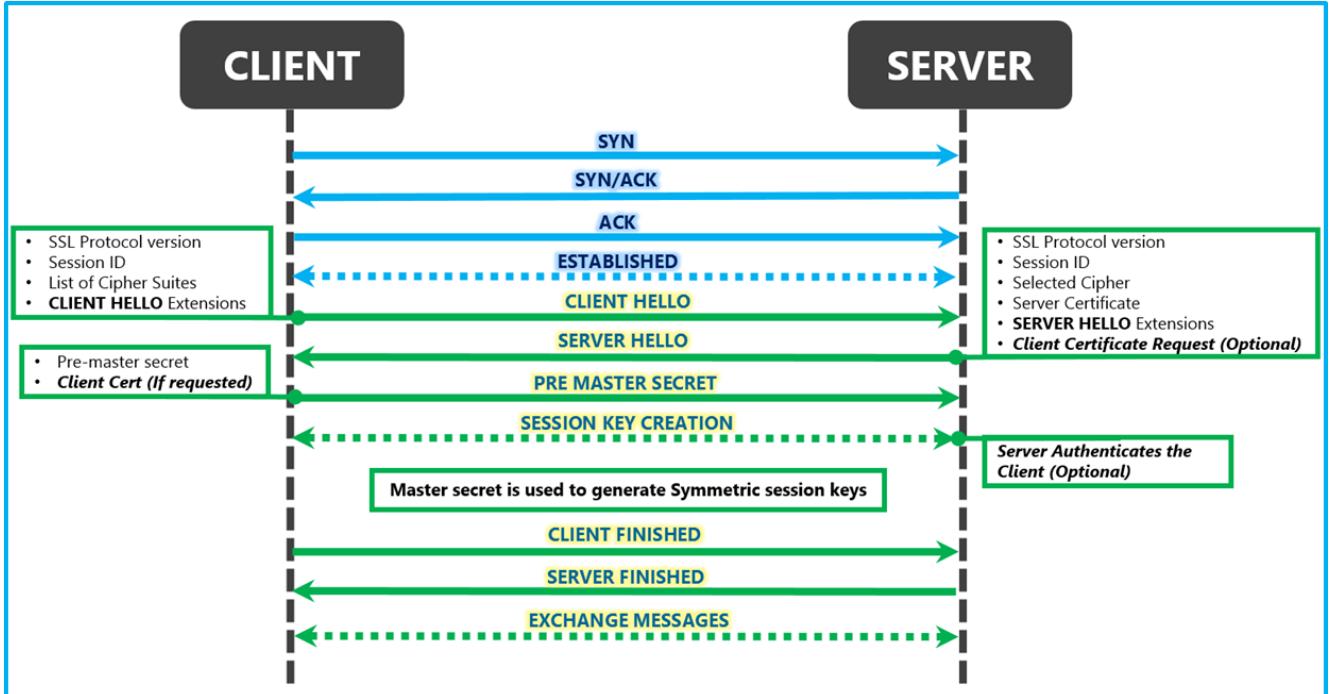


FIGURE 5.27 – Diagramme du déroulement du protocole TLS[19]

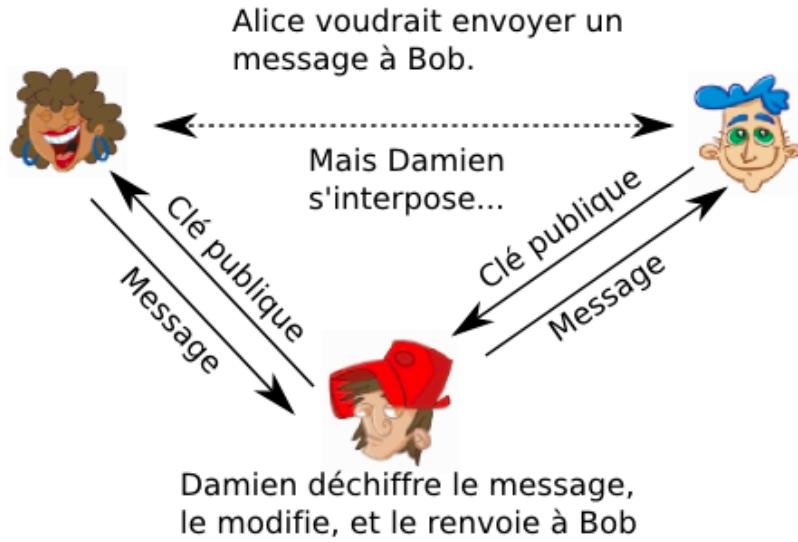


FIGURE 5.28 – Exemple d'une attaque de l'homme du milieu[17]

La figure 5.29 illustre où le protocole est déployé dans la solution implémentée dans ce mémoire. Veuillez noter que les certificats utilisés par la base de données et Grafana ont été autosignés. Ils ne protègent pas le client contre des attaques de l'homme du milieu, mais leur utilisation est gratuite. Ils peuvent donc être exploités pendant la phase de développement pour éviter des coûts non nécessaires. Lors du déploiement de la solution complète, un certificat signé par une tierce partie de confiance peut être acquis pour remplacer les certificats autosignés.

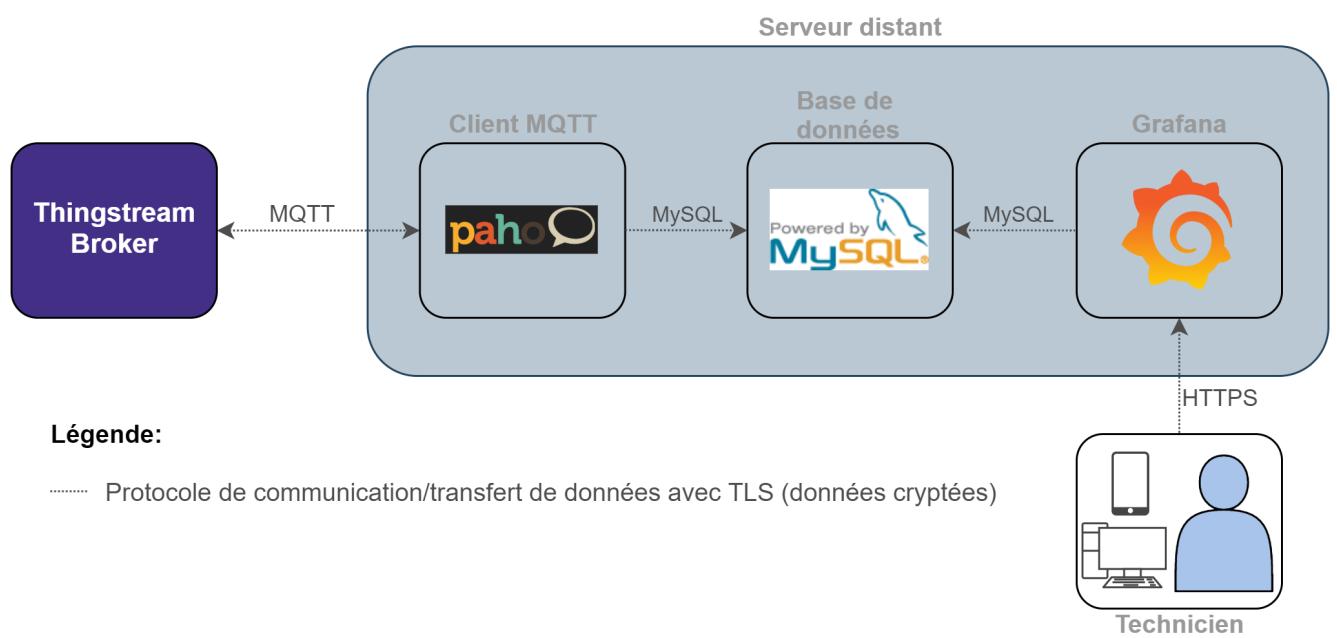


FIGURE 5.29 – Architecture finale de l'application tournant sur le serveur distant

Chapitre 6

Tests et résultats

L'intégralité du logiciel de monitoring et les composants matériels ont été testés ensemble afin d'évaluer la solution produite au cours de ce mémoire. Malheureusement, le prototype n'a pas pu être testé en conjonction avec l'infrastructure du CERHIS. À la place, les tests ont été réalisés dans un réseau domestique/*Home Area Network* (HAN). Les résultats obtenus ici peuvent ne pas refléter entièrement ceux qui seraient observés dans le réseau local de CERHIS. En effet, les deux réseaux présentent les différences suivantes :

- Le réseau domestique est caractérisé par une multitude de dispositifs de types divers tels que des smartphones, des décodeurs TV, des appareils de streaming sans fil et des ordinateurs. Le réseau de CERHIS se compose principalement de tablettes, du serveur, des points d'accès et potentiellement du contrôleur de charge solaire MPPT. Les différents dispositifs peuvent agir différemment lorsqu'ils reçoivent des requêtes PING ou ARP. De plus, les topologies distinctes peuvent avoir un impact sur le délai des réponses, ce qui peut aussi influencer le monitoring des appareils.
- Le réseau domestique ne possède aucun serveur similaire à celui utilisé par CERHIS. Un Raspberry Pi 3B a été employé pour jouer le rôle du serveur de CERHIS et les *node_exporter* et *process-exporter* ont été configurés dessus. En revanche, contrairement à la situation du serveur de CERHIS, presque aucun trafic de données dans le réseau domestique n'était dirigé vers ce Raspberry. Cela peut aussi avoir une incidence sur les résultats du test.
- La couverture et la qualité des réseaux mobiles en Belgique sont différentes de celles de la République Démocratique du Congo. Puisque le prototype n'utilise que le réseau 2G, la disparité ne devrait pas être frappante. Tout de même, une couverture moins bonne peut avoir un impact sur les capacités du prototype à se connecter au réseau et à transmettre des informations.
- Finalement, le nombre de dispositifs connectés dans le réseau domestique varie en fonction du moment de la journée. Cela est particulièrement vrai pour les smartphones qui se déconnectent lorsqu'ils ne se trouvent plus dans la maison. Le réseau de CERHIS est plus stable, les tablettes sont assignées au centre hospitalier et ne devraient pas quitter les lieux. Cela a un impact sur le nombre d'alertes produites au cours d'une journée, ce qui peut influencer le fonctionnement du système. (Principalement la taille de la base de données et la consommation énergétique)

Malgré ces différences, le réseau domestique offre tout de même la possibilité de mettre à l'épreuve le prototype. En effet, le test effectué sur cet environnement permet de vérifier qu'aucune erreur majeure n'a été commise dans la conception de ce dernier. En outre, un ensemble de métriques

surveillées est utilisé pour définir le comportement de base du prototype. Ces métriques seront particulièrement intéressantes à l'avenir pour comparer les résultats obtenus ici avec ceux d'un futur test réalisé avec l'infrastructure de CERHIS.

6.1 Définition de l'environnement

Avant de présenter les résultats, l'environnement et les conditions du test doivent être spécifiés afin que les résultats puissent être correctement interprétés. Le réseau domestique surveillé était composé des dispositifs suivants :

- Modem sans fin Compal CH7465LG-TN (Modem Telenet)
- 1 décodeur TV (Telenet)
- 1 boîtier multimédia Android
- 1 Google Chromecast
- 2 ordinateurs (Windows 10)
- 1 Access point (Telenet)
- 3 Smartphones (Android 10.0)
- 1 smartphone (Android 8.0)
- 1 Radio Internet
- 1 Raspberry Pi 3B+ (joue le rôle du serveur de CERHIS)

À cette liste s'ajoutent encore quelques dispositifs qui n'ont pas été monitorés, tels que deux ordinateurs portables supplémentaires. Ces machines ont été déconnectées du réseau avant que le mappage soit effectué. Elles n'ont pas été monitorées afin de vérifier si leurs connexions intermittentes affecteraient la surveillance des autres dispositifs. Ceci permet de simuler la situation où un technicien connecte momentanément son appareil au réseau local du centre hospitalier.

Les fichiers de configuration des *exporters* sont inclus avec le code développé au cours de ce mémoire. La plupart des paramètres ont été conservés par défaut. Un processus Python sans fin a aussi été lancé dans le Raspberry Pi 3B+ afin que le *process-exporter* puisse recueillir des informations dessus.

Ceci reprend l'entièreté de l'environnement du réseau domestique. Le client de monitoring utilisé correspond à celui qui a été développé au long de ce mémoire. Pour rappel, il est constitué par les composants suivants :

- Raspberry Pi 2B
- SIM800L + carte SIM Thingstream
- XL4015
- Le nouveau boîtier avec ventilateur

L'application de monitoring présenté dans la section 5.1 était exécutée sur ce Raspberry Pi 2B.

Pour déployer le serveur de la solution de monitoring, la Google Cloud Platform[14] a été utilisée. Une instance d'une machine virtuelle tournait le client MQTT et l'application de Grafana, et une instance MySQL a été employée pour la base de données. Il n'y a aucune raison particulière attachée à l'utilisation de cette plateforme au lieu de AWS ou Azure. Des instances dans les autres plateformes pourraient être exploitées avec le même effet. De façon similaire, un serveur privé pourrait également être employé pour déployer cette même application.

La figure 6.1 illustre la manière dont la solution a été déployée afin de réaliser ce test. Un service *node_exporter* a également été configuré sur le Raspberry Pi 2B exécutant le code du client de

monitoring. Cet exporter a permis de collecter de nombreuses métriques sur le fonctionnement de celui-ci, comme la température et l'utilisation du CPU. La section suivante reprend les résultats du test.

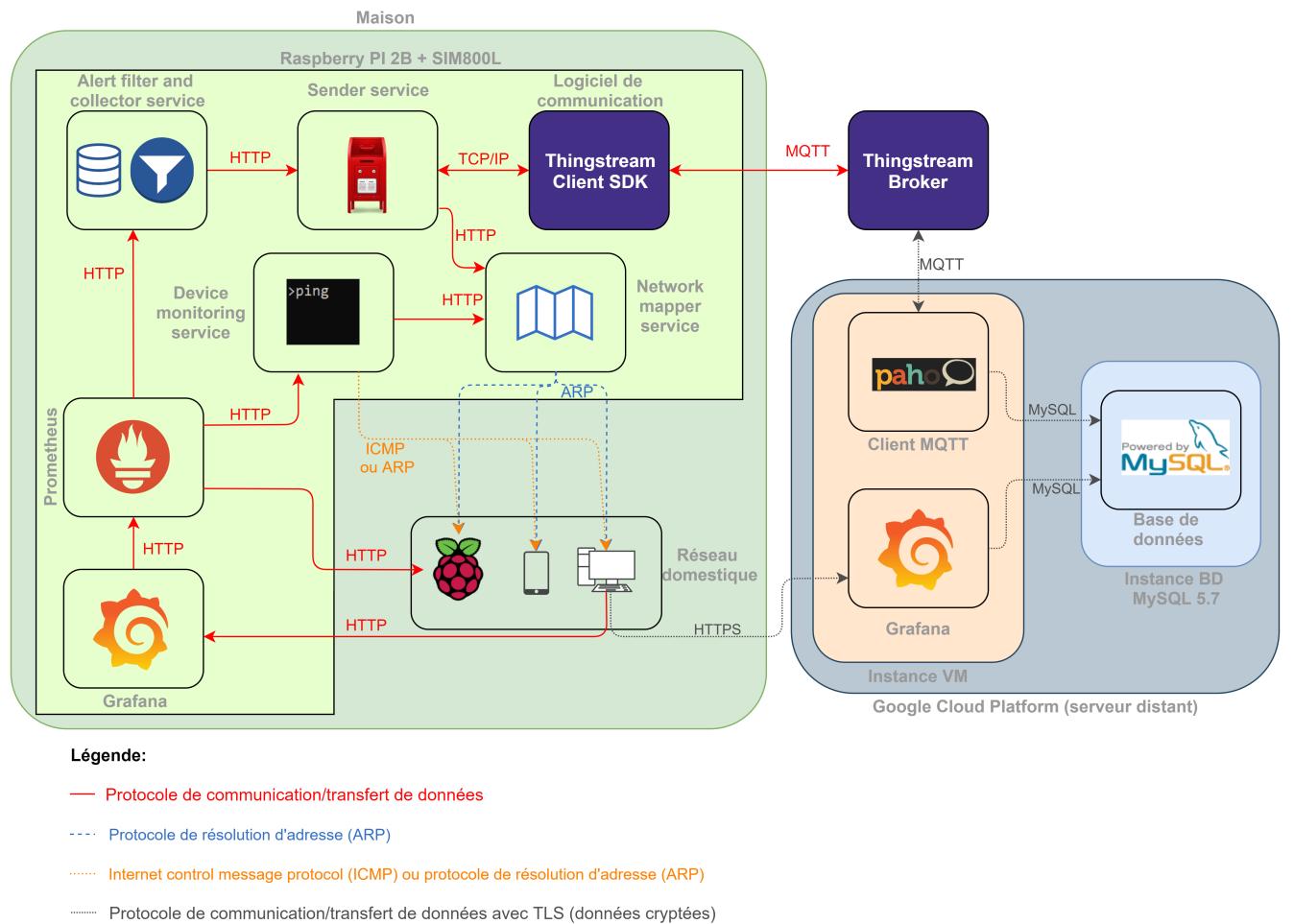


FIGURE 6.1 – Architecture complète de l'environnement de tests

6.2 Résultats

Le prototype ainsi que toute la solution de monitoring ont été testés pendant une période de 3 semaines et demie en continu. Cependant, la solution de communication a été éteinte après 6 jours (ceci sera expliqué plus tard). Malgré cela, la partie restante de la solution de monitoring a continué à fonctionner, seulement les avertissements par SMS et les données sur le serveur ont été impactées.

6.2.1 Consommation énergétique

La première métrique du prototype surveillée fût la consommation énergétique. Connaître sa consommation est important afin de correctement dimensionner la capacité de la batterie requise pour celui-ci. Le tableau 6.1 reprend les mesures qui ont été prises selon les situations suivantes :

- Fonctionnement normal (tous les services de monitoring en cours, tous les composants sont utilisés)
- Fonctionnement normal, mais sans ventilateur

- Fonctionnement normal, mais sans ventilateur et sans SIM800L (mais avec le XL4015)
- Seulement le Raspberry (tous les services toujours en cours, mais un chargeur de 5V a été directement connecté au Raspberry. Le step-down n'a donc pas été utilisé)

Situations	Min	Max	Moyenne
Normal	2.5W	2.6W	2.53W
Sans ventilateur	2.3W	2.3W	2.3W
Sans ventilateur et sans SIM800L	2.1W	2.2W	2.14W
Seulement le Raspberry Pi	1.6W	1.7W	1.66W

TABLE 6.1 – Consommation énergétique du prototype selon différentes situations

L'ensemble de ces mesures permet d'inférer plusieurs choses. Premièrement, l'efficacité du step-down utilisé est aux alentours de 76% (voir 6.1). Cette valeur est inférieure à celle documentée dans la fiche technique du XL4015 (85 – 90%). Cela est probablement dû à plusieurs raisons. En effet, les valeurs mesurées ici ne sont pas très précises ce qui peut influencer les résultats. En outre, un chargeur de 5V a dû être utilisé pour alimenter le Raspberry Pi lorsque sa consommation a été mesuré. En revanche, un chargeur de 12V est utilisé pour alimenter tout le prototype. Les différentes pertes au niveau des chargeurs influencent le résultat. Finalement, des composants de moins bonne qualité peuvent aussi provoquer une chute dans l'efficacité. Deuxièmement, ces mesures permettent d'établir le bilan de la consommation énergétique du prototype. Ce bilan est illustré dans la figure 6.2.

$$\eta = \frac{\text{énergie utile}}{\text{énergie totale}} = \frac{1.6W}{2.1W} = 0.76 \quad (6.1)$$

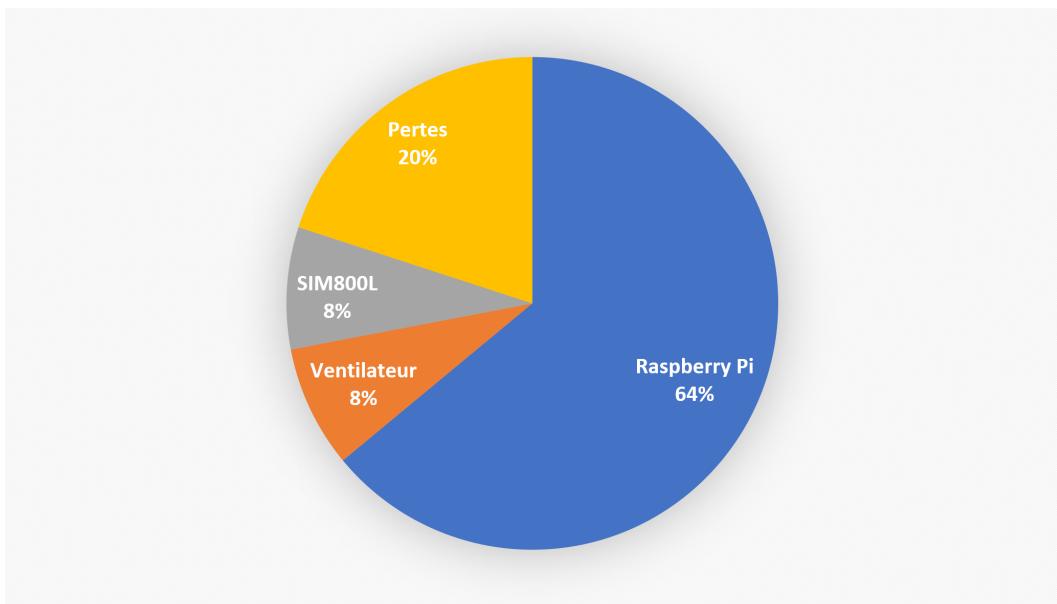


FIGURE 6.2 – Bilan énergétique du prototype

Une batterie de 12V et 6Ah devrait suffire pour alimenter le prototype pour un peu plus d'une journée. En revanche, ce temps pourrait être diminué si un technicien accède à l'application de Grafana qui tourne sur le Raspberry. En effet, pendant l'utilisation de ce logiciel, la consommation monte aux alentours de 3.5W pendant un court instant. Mesurer le vrai impact de ces piques de consommation n'a pas été possible car le taux de rafraîchissement du wattmètre était très faible. En outre, un phénomène assez similaire est attendu lorsque le SIM800L transmet des informations sur le réseau. Cependant, les limitations du wattmètre n'ont pas permis à nouveau d'obtenir des valeurs à ce sujet. De ce fait, plus de tests devront être réalisés dans le futur par rapport à ces deux cas.

6.2.2 Utilisation et température du CPU

Prometheus ainsi que tous les services du logiciel de monitoring se sont montrés peu gourmands en ressources du CPU du Raspberry Pi. Sur les 3 semaines d'exécution, l'utilisation du CPU n'a jamais dépassé les 10% (sur l'ensemble des 4 coeurs). L'utilisation moyenne était entre 1 et 2%. Le graphique de la figure 6.3 illustre l'évolution de l'utilisation du CPU au cours de la période de test.

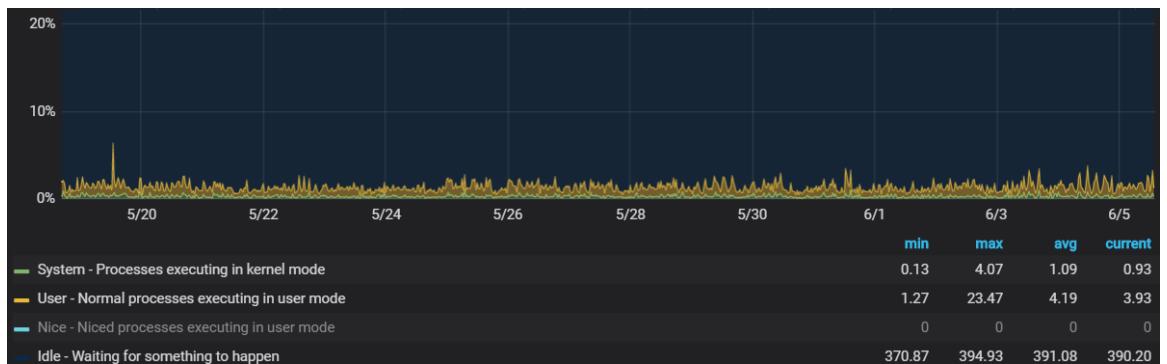


FIGURE 6.3 – Évolution de l'utilisation du CPU du Raspberry 2B. Les valeurs sur le tableau sont comprises entre 0 et 400 pour cent^a

^a= Le Raspberry Pi a quatre coeurs et l'utilisation de chaque cœur est mesurée entre 0 à 100 pour cent.
La valeur sur le tableau correspond à l'addition de la valeur d'utilisation des quatre coeurs.

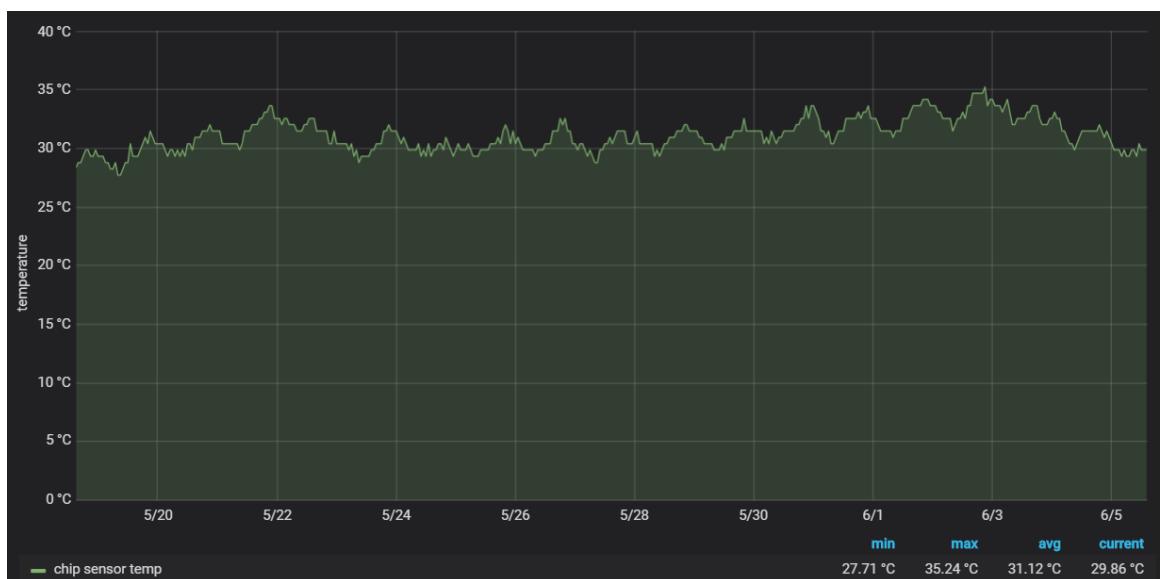


FIGURE 6.4 – Évolution de la température du CPU du Raspberry 2B.

L'évolution de la température du CPU est reprise sur la figure 6.4. Tout comme dans les tests réalisés précédemment (voir section 4.2.4), le système de refroidissement du nouveau boîtier contrôle correctement la température du CPU. Il y a une légère augmentation au début du mois de juin, qui atteint son pic aux alentours du troisième jour. Cela est lié à une augmentation de la température ambiante.

6.2.3 Détection d'alertes et envoi des messages

Cette partie était la plus complexe à tester. En effet, il est irréaliste de vérifier manuellement l'état de tous les dispositifs du réseau de façon continue sur une période aussi longue. Cela crée un problème puisqu'il n'y a pas un ensemble de données avec lesquel il est possible de comparer ce qui a été mesuré par le prototype. La stratégie pour examiner le fonctionnement de ce composant a dû être modifiée.

En revanche, il est possible de vérifier manuellement si un message d'alerte envoyée par le prototype correspond aux conditions actuelles de l'infrastructure. C'est donc cette approche qui a été prise. Tous les SMS expédiés par le prototype ont été vérifiés ainsi que les messages transmis au serveur. Pendant les 6 jours où la solution de communication est restée connectée, sur 70 alertes envoyées, une seule s'est révélée être un faux positif. Cette alerte était liée à la disparition de la radio Internet du réseau local, ce qui n'est jamais arrivé. Les données sur le serveur distant permettent de voir qu'à peine trois minutes après sa déconnexion, la radio Internet était à nouveau marquée comme étant présente (voir figure 6.5).

Liste des alertes de type Device Down					
node	mac_address	device_type	status	time	end
192.168.0.183	00:22:61:E4:F1:24	IP RADIO	resolved	2020-05-16 00:07:31	2020-05-16 00:10:31

FIGURE 6.5 – Données de l'alerte qui a été incorrectement signalée

Cette alerte pourrait être liée à une déconnexion momentanée de la radio, mais cela ne peut pas être dit avec certitude. De ce fait, l'erreur est attribuée au prototype. Tout de même, un faux positif sur 70 alertes envoyées correspond à un taux d'erreur de moins de 2%, ce qui est très faible. En plus de vérifier la présence de faux positifs parmi les alertes, ce test a également permis d'identifier les problèmes suivants :

- Lorsqu'une alerte était en cours et qu'une deuxième était détectée, deux SMS étaient expédiés au technicien malgré le fait qu'un SMS ait déjà été envoyé pour la première alerte. Par exemple, supposons que trois appareils sont déconnectés depuis une heure et que le technicien ait déjà été averti à propos de leur disparition. Si un quatrième dispositif se déconnecte, quatre SMS étaient transmis au technicien, un par appareil absent. Ceci était dû au fait que Prometheus *push* une liste contenant l'entièreté des alertes en cours chaque fois qu'une nouvelle alerte est identifiée. Ce problème a été résolu en ajoutant un filtre dans le service *Alert filter and collector*. Ce dernier vérifie si un message a déjà été envoyé dans le passé et, si c'est le cas, il ne le *push* pas afin d'éviter un SPAM d'avertissements.
- Les données de Prometheus étaient horodatées en fonction du fuseau horaire UTC. En revanche, les données du service *network mapper* étaient horodatées selon le fuseau horaire local. Cela créait une incohérence dans les informations. Ce problème a également été corrigé en horodatant les données du *network mapper* selon le fuseau horaire UTC.

- Certains smartphones Android éteignent le Wi-Fi pendant la nuit dans le but d'économiser de la batterie. Cela s'est produit avec deux des quatre smartphones surveillés. Ce comportement peut généralement être désactivé dans le système d'exploitation. Une autre solution pourrait être de faire taire les alertes de présence pendant la nuit. Cependant, ceci n'a pas été implémenté dans le service de device monitoring.
- Le dernier souci détecté est lié à un bug dans le SDK de Thingstream. Le protocole MQTT stipule que si un client et le *broker* n'ont pas communiqué pendant une certaine période, les deux parties doivent échanger un PING pour vérifier que la connexion est toujours active. La durée de cette période est définie à l'aide de la variable *KEEP ALIVE* et elle est configurable. Cependant, le SDK de Thingstream impose que ce temps soit de 5 minutes, indépendamment de la configuration passée. Cela pose un problème puisque chaque PING requiert deux messages qui sont chargés par Thingstream. Sur un mois, cela correspond à un échange de plus de 17000 messages payants rien que pour les PINGs. Ceci se traduira par un coût allant de quatre à presque neuf dollars selon le tarif choisi (voir image 6.6). Après avoir contacté le support technique de Thingstream, ils ont reconnu que c'était une erreur de leur part et que le SDK allait être révisé à l'avenir. Malheureusement, ils n'ont pas donné une date et au temps de l'écriture de ce mémoire, le correctif n'est toujours pas disponible. Cela est la raison pour laquelle le service de connexion a été déconnecté. En quelques jours, le client avait largement dépassé son quota de messages et continuera à faire de même s'il n'était pas arrêté.

Name	Price (\$)	Unit	Description
MQTT Anywhere 12K	\$2.00	Block of MQTT message(s)	\$2 per 12K MQTT-SN messages per month.
MQTT Anywhere 5K	\$1.00	Block of MQTT message(s)	\$1 per 5K MQTT-SN messages per month.
MQTT Anywhere 2K	\$0.50	Block of MQTT message(s)	\$0.50 per 2K MQTT-SN messages per month.
MQTT Anywhere 500	\$0.25	Block of MQTT message(s)	\$0.25 per 500 MQTT-SN messages per month.

MQTT Anywhere plans allow SIM based Things to send [MQTT-SN messages](#) to the Thingstream broker. Each plan comprises a number of messages (the message block) for a given price per Thing per month.

Overage

Should a Thing enrolled on a plan exceed the maximum number of messages included in the message block for that plan, then an additional message block will be allocated to that Thing and an additional charge for the message block will be made. For example, a Thing enrolled on the MQTT Anywhere 5k plan sends 8,000 messages in a calendar month. The charge for this will be \$2.00 (\$1.00 for the original 5,000 message block included with the plan plus an additional charge of \$1.00 for the additional message block of 5,000 messages).

FIGURE 6.6 – Prix des différents tarifs de Thingstream.

6.2.4 Rapidité de la détection d'alertes

La dernière métrique surveillée est la durée requise par le prototype pour détecter et prévenir un technicien à propos de la disparition d'un dispositif. Ce test est facile à effectuer puisqu'il suffit de manuellement déconnecter un appareil du réseau et de compter le temps passé jusqu'à la réception du SMS. En revanche, ce temps n'est pas constant. Le service *device monitoring* ne contrôle la présence des dispositifs qu'une fois par minute. Le temps d'envoi d'un SMS sera plus ou moins long en fonction du délai passé depuis la dernière vérification. De ce fait, les résultats présentés sur le tableau 6.2 correspondent à une moyenne réalisée sur 10 tests, permettant de mieux visualiser quel sera le temps de réponse moyen du prototype.

Mesures	Temps moyen (s)	Temps max (s)	Temps min (s)
Détection dispositif disparu	56	78	41
Délai configuration Prometheus	120	120	120
Envoi du SMS	45	49	38
Total	221	243	206

TABLE 6.2 – Temps requis pour le prototype pour identifier, générer et envoyer une alerte à propos d'un dispositif qui s'est déconnecté du réseau (Les valeurs des colonnes max et min ne correspondent pas forcément au même test.)

Veuillez noter qu'en raison de la configuration actuelle, Prometheus attend toujours deux minutes avant de déclencher une alerte. Ce temps peut être réduit pour obtenir un message plus rapidement, mais cela peut augmenter le nombre de faux positifs. En analysant les données recueillies par Prometheus de plus près, il est possible de constater qu'il faut généralement entre 41 à 78 secondes pour détecter qu'un dispositif est absent du réseau (service *device monitoring*). Ensuite, une fois que les deux minutes se sont écoulées au niveau de Prometheus, entre 38 à 49 secondes sont requises pour que le SMS arrive chez le technicien. Ceci est lié au fait que les messages sont ajoutés dans une file d'attente dans le client de Thingstream. Le SDK retire et traite un message de la file une fois toutes les 30 secondes. Un processus assez similaire a lieu dans le *sender service*, qui patiente une dizaine de secondes avant de tout transmettre au client de Thingstream. Cela permet d'attendre pour vérifier si d'autres alertes doivent également être envoyées. De ce fait, en cas d'alertes multiples, les SMS sont expédiés en premier lieu au client de communication. Cela évite la situation où un SMS d'alerte serait trop retardé à cause de messages à transmettre au serveur distant, car la file d'attente fonctionne sur le principe premier entré, premier sorti (FIFO).

Finalement, il est possible de conclure que le temps entre le début d'un événement (disparition d'une tablette ou alerte dans le serveur) et la réception d'un SMS sera au maximum de 5 minutes. Cette valeur paraît tout à fait raisonnable, mais pourrait encore être optimisée dans le futur. En effet, avec des tests sur l'infrastructure de CERHIS il serait possible de configurer les différents délais afin d'avoir le meilleur compromis entre rapidité, consommation énergétique et fiabilité des alertes.

Chapitre 7

Conclusion

Au cours de ce mémoire de fin d'études, une solution de monitoring pour le système d'information hospitalier CERHIS a été développée. Pour ce faire, les éléments matériels à utiliser devaient être établis en tenant compte des particularités de l'environnement. En effet, le prototype présenté ici est prêt à fonctionner dans des milieux avec des températures élevées et sans accès à une connexion Internet filaire. Le premier problème a été abordé avec le développement d'un boîtier spécifiquement conçu pour abriter les différents composants électroniques. Parmi ces composants figure un Raspberry Pi, qui fournit toute la puissance de calcul au prototype. Pour solutionner le second, le choix s'est porté sur l'utilisation d'un module SIM800L et la plateforme de Thingstream. Les tests réalisés sur ce prototype ont mis en évidence les bonnes capacités de refroidissement de celui-ci. En outre, les problèmes de communication rencontrés avec le SIM800L lors du projet précédent ont été résolus. Cela a permis d'exploiter le SIM800L de manière très fiable.

Le logiciel de monitoring développé est divisé en deux grandes parties. Premièrement, il y a le client qui tournera sur le Raspberry Pi dans le centre hospitalier. Cette application est basée sur une architecture de microservices et emploie le logiciel de Prometheus pour surveiller et stocker toutes les informations relatives au monitoring des dispositifs. L'outil de visualisation de données Grafana est couplé avec Prometheus pour donner aux techniciens un aperçu de l'état de l'infrastructure de CERHIS. Mais tout cela n'a pas été réalisé sans faute. Une première approche utilisant le protocole SNMP a été tentée. Cette solution s'est révélée efficace, mais elle n'était pas évolutive, ce qui causerait d'énormes problèmes dans l'avenir. Ceci constitue la plus grosse erreur qui a été commise lors de ce mémoire, puisqu'une recherche plus approfondie aurait montré que d'autres solutions plus performantes étaient disponibles sur le marché. Heureusement, cette erreur a pu être corrigée avec l'utilisation de Prometheus. Toutes les alertes détectées par le logiciel Prometheus sont transmises par SMS aux techniciens et vers un serveur distant.

La deuxième partie de ce logiciel de monitoring consiste dans l'application tournant sur un serveur distant. Elle centralise toutes les données envoyées par les clients et les rend accessibles aux techniciens, même lorsqu'ils ne se trouvent pas dans le centre hospitalier. Le système de communication utilisé ne permet pas que toutes les métriques collectées par le client puissent être transmises sur ce serveur. Cependant, toutes les alertes y sont envoyées, incluant des mises à jour sur celles-ci. L'outil Grafana a été employé pour offrir une visualisation des données du serveur.

Les tests effectués sur l'intégrité de la solution développée ont été très prometteurs. Malgré le fait qu'ils n'ont pas été réalisés en conjonction avec l'infrastructure de CERHIS, ils démontrent les capacités du prototype, en particulier celles de l'application de monitoring. Cependant, tout n'a pas été sans failles. Le SDK de Thingstream a présenté un bug qui a un grand impact sur l'utilisation de cette plateforme. Le support technique de Thingstream a assuré qu'il sera corrigé

à l'avenir, mais aucune date n'a été donnée quant au moment où cela pourrait se produire.

7.1 Suggestions d'amélioration

La solution de communication avec Thingstream s'est montrée comme étant le goulet d'étranglement de ce prototype. Malgré sa fiabilité irréprochable, son architecture opère de manière différente que celle du logiciel de monitoring (sauf pour les alertes). En outre, la plateforme de Thingstream n'a pas été conçue pour transmettre le volume de données qui est acquis par Prometheus. Dans l'avenir, une étude locale, c'est-à-dire en République Démocratique du Congo, doit être réalisée dans le dessein d'établir quelle est la solution de communication la plus adaptée pour ce projet. En effet, Thingstream offre une grande fiabilité au détriment de la quantité d'informations qui peut être communiquée. Les deux autres solutions proposées (Routeur LTE et le protocole point à point) peuvent résoudre le problème lié au volume de données, mais pourraient avoir un impact sur la fiabilité. C'est afin de comprendre ce dernier point que l'étude doit s'orienter, c'est-à-dire trouver la solution proposant le meilleur compromis entre coût, fiabilité et quantité de données qu'elle peut transmettre. Un deuxième point qui a encore besoin d'un peu plus de développement est le circuit imprimé employé avec ce prototype. Le circuit imprimé réalisé au cours de l'année précédente a été réutilisé cette année, mais il n'est plus adapté au nouveau prototype. En effet, avec l'addition d'un ventilateur, il serait tout à fait intéressant de pouvoir connecter ce dernier au Raspberry Pi dans le but de contrôler sa vitesse. Cela fournira une manière très simple de diminuer la consommation énergétique du prototype sans avoir un impact sur les capacités de refroidissement du boîtier. En outre, cela permettra aussi de surveiller l'état du ventilateur. Dernièrement, cette solution doit impérativement être testée dans l'environnement de CERHIS. Sans cela, les différents paramètres ne pourront pas être optimisés et des erreurs potentielles pourraient passer inaperçues.

Annexe A

Tableau comparatif des réseaux

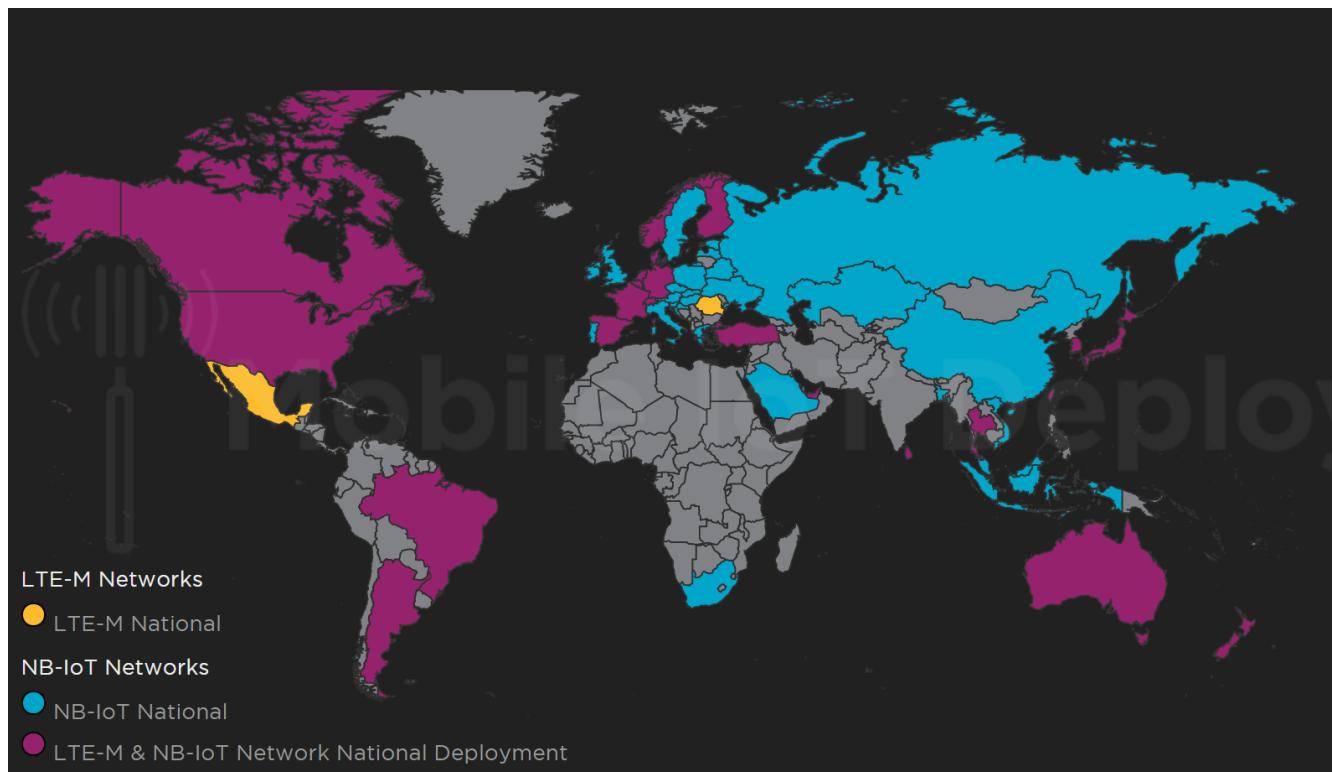


FIGURE A.1 – Couverture des réseaux NB-IoT et LTE-M [15]

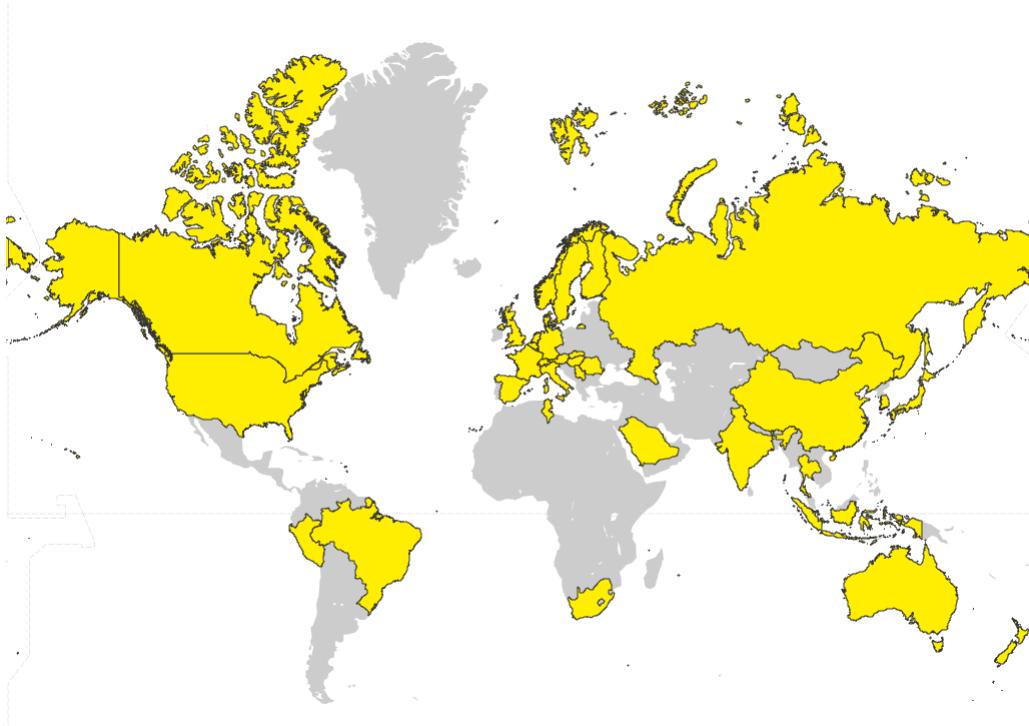


FIGURE A.2 – Couverture du réseau public LoRaWAN [18]

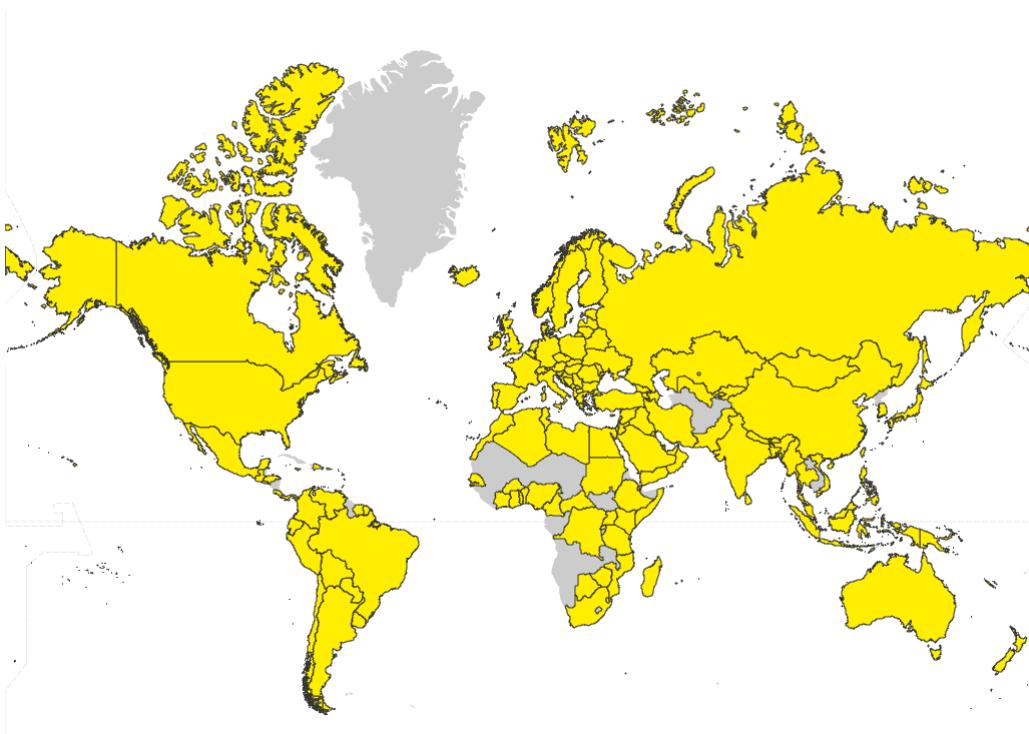


FIGURE A.3 – Couverture des réseaux public et de développement LoRaWAN [18]

	LoRaWAN	Réseaux mobiles (2G / 3G / 4G / 5G)	Sigfox	Satellite	Wi-Fi	Zigbee	Bluetooth	Réseaux mobiles IoT (NB-IoT, LTE-M)
Débit	0.3 - 50 kbps	EDGE : 384 Kbps HSPA+ : 42 Mbps LTE-A : 1 Gbps 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 LTE-M : 1 Mbps
Bande de fréquences	868 - 915 MHz	400 - 3000 MHz + 25 - 39 Ghz (5G)	862 - 928 MHz	1 - 40 GHz	2.4 GHz / 5 GHz 2400 MHz	868/915/ 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 LTE-M : 1 - 11 km
Couverture	Figures A.2 et A.3	Mondiale	Europe de l'ouest principalement	Mondiale	Réseau privé	Réseau privé	Réseau privé	Figure A.1
Sécurisé	Oui (AES 128)	Exploits possibles dans le réseau GSM. 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)

TABLE A.1 – Tableau comparatif des différentes solutions de communication. (Basé sur [16, 60, 46, 64])

Annexe 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)

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

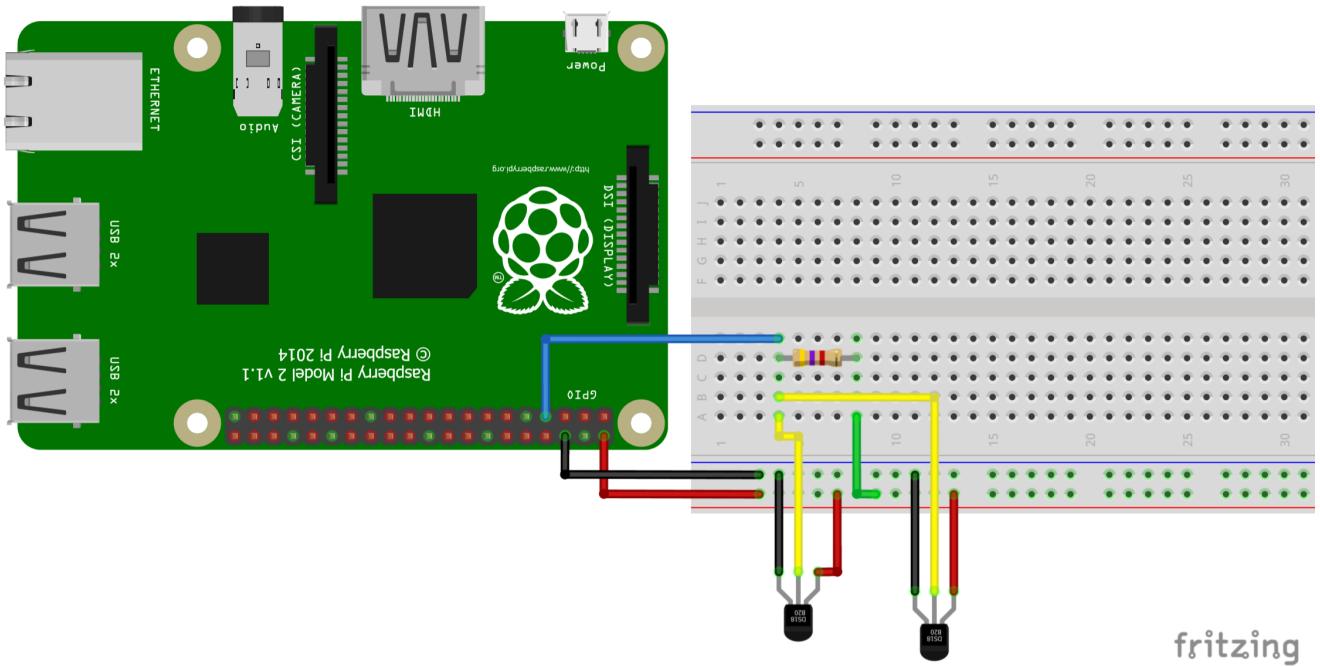


FIGURE B.1 – Schéma des connexions des capteurs de température au Raspberry serveur

- -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 *server_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 coeurs 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.

Annexe C

Choix de la plateforme (Rapport du projet 2018-2019)

3.1.1 Choix de la plateforme

La plateforme est le composant le plus important de ce projet. En effet, son choix a des répercussions sur l'entièreté de celui-ci. La plateforme influence le choix des autres éléments, le code à produire, le système d'alimentation et le coût. Bien choisir ce composant est donc primordial pour assurer la réussite du projet.

Lors des deux projets précédents, deux plateformes très différentes ont été choisies. Comme expliqué dans la section 1.3, le projet de monitoring s'est appuyé sur un smartphone Android, tandis que le projet de dimensionnement a utilisé un Arduino MKR 1400 GSM. Cependant, lors de son mémoire, Martin Delobbe a trouvé un problème avec le système d'exploitation Android. De façon à diminuer la consommation énergétique du smartphone, Android possède un mode appelé Doze mode qui empêche les applications de tourner en arrière-plan pendant les moments d'inactivité du smartphone. [9] Pour pallier ce problème, Martin suggère le remplacement du smartphone par une Raspberry Pi. Les trois plateformes, Raspberry Pi, Arduino et smartphone, sont comparés selon différents critères dans le tableau 1.

	Arduino MKR 1400 GSM	Smartphone	Raspberry Pi
Prix	88,86 €[RS]	≈ 90 €[2]	≤ 40 €[RS]
Envoi de SMS	Oui	Oui	Nécessite un module
Consommation énergétique	16mA [16]	≤ 500mA [22]	≤ 400mA [10, 15]
Puissance de calcul	Faible	Élevée	Élevée
Modularité	Bonne	Mauvaise	Bonne

TABLE 1: Comparaison entre les différentes plateformes

Après une analyse approfondie des trois options, il ressort que la Raspberry Pi est effectivement la meilleure plateforme pour réaliser ce projet. En effet, l'Arduino ne possède pas une puissance de calcul suffisante pour réaliser toutes les tâches demandées et le smartphone n'est plus une solution viable pour ce projet. Toutefois, il existe plusieurs modèles différents de la Raspberry Pi sur le marché. De ce fait, il faut également trancher entre les différentes options disponibles et trouver le modèle le plus adéquat pour ce projet.

	Raspberry Pi Zéro WH	Raspberry Pi 2B	Raspberry Pi 3B	Raspberry Pi 3B+
Prix	19,90 €[LDLC]	36,53 €[RS]	35,66 €[RS]	35,66 €[RS]
Consommation énergétique (IDLE) ⁵	120mA	230mA	230mA	400mA
Puissance de calcul	Suffisante	Élevée	Élevée	Élevée
Connectivité	Wi-Fi	Ethernet	Wi-Fi+Ethernet	Wi-Fi+Ethernet

TABLE 2: Comparaison entre les différentes Raspberry Pi disponibles

Finalement, puisqu'un port Ethernet était nécessaire pour connecter la Raspberry Pi à un routeur à Goma, la Raspberry Pi 2B a été choisie. Selon [15], la consommation de ce modèle semble quand même être légèrement inférieure à celle de la Raspberry Pi 3B.⁴ Cependant, ce dernier modèle reste tout de même très intéressant du fait qu'il peut également utiliser des réseaux Wi-Fi.

Le modèle 3B+ possède une consommation presque deux fois plus élevée que le modèle précédent sans apporter aucun réel avantage à ce projet. De ce fait, il s'avère que ce modèle n'est pas un bon choix. Dernièrement, le modèle Zéro WH peut se révéler très intéressant si un port Ethernet n'est plus nécessaire.

Bibliographie

- [1] 5a 180khz 36v buck dc to dc converter xl4015 : Datasheet. <http://www.xlsemi.com/datasheet/XL4015%20datasheet.pdf>. [Accédé le 14-Mai-2020].
- [2] Aedes : Qui sommes nous ? <http://www.aedes.be/fr/presentation.html>.
- [3] Africa's talking : Ussd. <https://africastalking.com/ussd>. [Accédé le 14-Mai-2020].
- [4] Android developers doc : Optimize for doze and app standby. <https://developer.android.com/training/monitoring-device-state/doze-standby>. [Accédé le 3-Juin-2020].
- [5] 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].
- [6] 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].
- [7] 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].
- [8] Cellule de coopération au développement : Mission. <https://polytech.ulb.be/fr/international/cellule-de-cooperation-au-developpement/mission>.
- [9] Data flow manager : Flow-based visual programming. <https://thingstream.io/products/data-flow-manager/>. [Accédé le 12-Mai-2020].
- [10] 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].
- [11] Fing : Connected devices recognition based on network protocols. <https://www.fing.com/business>. [Accédé le 5-Juin-2020].
- [12] Github : Exporter for machine metrics (node exporter repository). https://github.com/prometheus/node_exporter. [Accédé le 30-Mai-2020].
- [13] Github : Prometheus exporter that mines /proc to report on selected processes (process exporter repository). <https://github.com/ncabatoff/process-exporter>. [Accédé le 30-Mai-2020].
- [14] Google cloud platform. <https://cloud.google.com/?hl=fr>. [Accédé le 5-Juin-2020].
- [15] Internet of things : Deployment map. <https://www.gsma.com/iot/deployment-map/>. [Accédé le 19-Mai-2020].
- [16] Iot connectivity comparison (gsm vs lora vs sigfox vs nb-iot). <https://www.polymorph.co.za/iot-connectivity-comparison-gsm-vs-lora-vs-sigfox-vs-nb-iot/>. [Accédé le 28-Avril-2020].

- [17] L'attaque de l'homme du milieu, et les certificats électroniques. <http://www.bibmath.net/crypto/index.php?action=affiche&quoi=moderne/certificat>. [Accédé le 2-Juin-2020].
- [18] Lorawan : Coverage & operator maps. <https://lora-alliance.org/>. [Accédé le 19-Mai-2020].
- [19] Microsoft : Ssl handshake and https bindings on iis. <https://docs.microsoft.com/en-us/archive/blogs/kaushal/ssl-handshake-and-https-bindings-on-iis>. [Accédé le 2-Juin-2020].
- [20] 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].
- [21] Mqtt : Frequently asked questions. <http://mqtt.org/faq>. [Accédé le 14-Mai-2020].
- [22] Prometheus : Exporters and integrations. <https://prometheus.io/docs/instrumenting/exporters/>. [Accédé le 30-Mai-2020].
- [23] Raspberry pi 2 model b. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. [Accédé le 7-Mai-2020].
- [24] Red hat : Que sont les microservices ? <https://www.redhat.com/fr/topics/microservices/what-are-microservices>. [Accédé le 24-Mai-2020].
- [25] Renouvellement du réseau mobile. <https://www.swisscom.ch/fr/about/entreprise/portrait/reseau/remplacement-2g.html>. [Accédé le 28-Avril-2020].
- [26] Tableau : Guide pratique de la visualisation de données : définition, exemples et ressources didactiques. <https://www.tableau.com/fr-fr/learn/articles/data-visualization>. [Accédé le 19-Mai-2020].
- [27] 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].
- [28] 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].
- [29] Ulb-coopération : Vision, missions et valeurs. <https://www.ulb-cooperation.org/fr/vision-missions-et-valeurs>.
- [30] Zabbix documentation 4.4 : Database creation. https://www.zabbix.com/documentation/current/manual/appendix/install/db_scripts. [Accédé le 29-Avril-2020].
- [31] Ch7 :address resolution protocol(arp). [https://gsephrioth.github.io/Ch7-Address-Resolution-Protocol\(ARP\)/](https://gsephrioth.github.io/Ch7-Address-Resolution-Protocol(ARP)/), Apr. 2017. [Accédé le 30-Mai-2020].
- [32] 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].
- [33] Hivemq : Mqtt topics & best practices - mqtt essentials : Part 5. <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/>, Aug. 2019. [Accédé le 24-Mai-2020].
- [34] A Banks, E Briggs, K Borgendale, and R Gupta. Mqtt version 5.0. *OASIS Standard*, 2019.
- [35] 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].
- [36] Nikos Bikakis. Big data visualization tools. *arXiv preprint arXiv:1801.08336*, 2018.

- [37] B. Brazil. *Prometheus - Up and Running : Infrastructure and Application Performance Monitoring*. O'Reilly Media, 2018.
- [38] Brian Brazil. Evolving from machines to service. <https://grafana.com/blog/2016/01/12/evolving-from-machines-to-services/>, Jan. 2016. [Accédé le 30-Mai-2020].
- [39] 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>.
- [40] 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>.
- [41] Giuseppe Cattaneo, Giancarlo Maio, and Umberto Petrillo. Security issues and attacks on the gsm standard : a review. *JOURNAL OF UNIVERSAL COMPUTER SCIENCE*, 19 :2437–2452, 01 2013.
- [42] Cisco. Cisco annual internet report (2018–2023). Technical report, Cisco, 03 2020.
- [43] Martin Delobbe. Conception et réalisation d'une solution permettant la centralisation et la visualisation de données de monitoring provenant de centres de santé en rdc. Master's thesis, 2018.
- [44] 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.
- [45] 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-a-assaut-de-l-internet-des-objets>, Dec 2016. [Accédé le 28-Avril-2020].
- [46] Brandon Foubert and Nathalie Mitton. Long-range wireless radio technologies : A survey. *Future Internet*, 12 :13, 01 2020.
- [47] Ethan Galsta. Ndutils documentation version 2.x. <https://assets.nagios.com/downloads/nagioscore/docs/ndutils/NDOUTils.pdf>, Mar 2017. [Accédé le 29-Avril-2020].
- [48] Marcin Hajdul and Arkadiusz Kawa. Global logistics tracking and tracing in fleet management. pages 191–199, 03 2015.
- [49] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *ACM Queue*, 8 :20, 01 2010.
- [50] A. Kirk. *Data Visualisation : A Handbook for Data Driven Design*. SAGE Publications, 2019.
- [51] 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].
- [52] 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.
- [53] 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].
- [54] 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.

- [55] Wilbroad Mutale, , Namwinga Chintu, Cheryl Amoroso, Koku Awoonor-Williams, James Phillips, Colin Baynes, Cathy Michel, Angela Taylor, and Kenneth Sherr. Improving health information systems for decision making across five sub-saharan african countries : Implementation strategies from the african health initiative. *BMC Health Services Research*, 13(S2), May 2013.
- [56] 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].
- [57] World Health Organization et al. *Monitoring the building blocks of health systems : a handbook of indicators and their measurement strategies*. World Health Organization, 2010.
- [58] 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.
- [59] Stylianos Rigas. Monitoring a multi-cluster environment using prometheus federation and grafana. <https://mattermost.com/blog/monitoring-a-multi-cluster-environment-using-prometheus-federation-and-grafana/>, Feb. 2020. [Accédé le 30-Mai-2020].
- [60] Ramon Sanchez-Iborra and Maria-Dolores Cano. State of the art in lp-wan solutions for industrial iot services. *Sensors*, 16(5) :708, 2016.
- [61] 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].
- [62] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, 1 :2, 2013.
- [63] 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].
- [64] 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.
- [65] 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].