

ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE NANTES

DÉPARTEMENT D'INFORMATIQUE

RAPPORT DE RECHERCHE ET DÉVELOPPEMENT

Rapport final

Collecte automatique de données

Pierre-Adrien DELISLE & Samuele DA SILVA

04 février 2020

encadré par Benoit PARREIN & Julien VANDAELE

— Équipe RIO —

LABORATOIRE DES SCIENCES DU NUMÉRIQUES DE NANTES

INRIA

coordinateur : Philippe LERAY



UNIVERSITÉ DE NANTES

Avertissement

Toute reproduction, même partielle, par quelque procédé que ce soit, est interdite sans autorisation préalable.

Une copie par xérographie, photographie, photocopie, film, support magnétique ou autre, constitue une contrefaçon passible des peines prévues par la loi.

Rapport final

Résumé

La plateforme FIT (Future Internet Testing facility) IoT lab, créée pour tester différents aspects importants pour les thématiques qui entourent l'Internet des objets, est constituée de 6 sites, répartis en France, qui contiennent de nombreux nœuds capteurs, et permettent de nombreuses expérimentations.

Le problème, très technique, est ici de collecter automatiquement, à intervalles réguliers dans la journée, différentes données sur des puces M3, qui constituent la majorité du parc de la plateforme FIT Lilloise, telles que la luminosité ou la température. Les objectifs pour arriver à une solution sont donc les suivantes :

1. Développement et Implémentation d'un firmware pour les nœuds m3, permettant de récupérer les données et de les pousser sur lien série ;
2. Développement d'un script python qui crée les expériences nécessaires, récupère les données poussées par les nœuds, et les pousse dans une base temporelle ;
3. Récupération, stockage et visualisation des données ;

Termes généraux : Internet des objets, FIT IoT, Données, Acquisition, Représentation, Stockage, Radio.

Mots-clés additionnels et phrases : Captation automatique, Nœuds m3, Lille, protocole, RPL, perturbations

Collecte automatique de données

Remerciements

Nous remercions Bastien Confais pour sa rigueur et son investissement dans notre travail, Benoit Parrein pour le suivi et les retours très rapides sur le travail effectué, Julien Vandaele pour sa disponibilité, ses explications très claires et ses retours très rapides à nos questions

Table des matières

1	Introduction	7
1.1	Sujet d'étude	7
1.2	Présentation de la problématique	7
1.3	Objectifs poursuivis	8
1.4	Travail à réaliser et travail réalisé	8
1.5	Plan de l'étude	9
2	État de l'art	11
2.1	Plateforme Fit IoT Lab	11
2.1.1	Présentation	11
2.1.2	Architecture IoT Lab	11
2.1.3	Tutoriels essentiels	13
2.2	Firmwares	14
2.2.1	Contiki	14
2.2.2	RIOT	15
2.2.3	Analyse	16
2.3	Base de données temporelle	16
2.3.1	Présentation	16
2.3.2	Analyse	16
2.4	Automatisation via Linux	18
2.4.1	Cron	18
2.4.2	Systemd	19
2.4.3	Comparaison	19
2.5	Protocole RPL	19
2.6	Métriques réseau sur le bruit et qualité	20
2.6.1	R-ED - Reciever ED	20
2.6.2	LQI : Link Quality Indicator	20

2.6.3	ETX : Expected transmission count	20
2.6.4	RSSI : Received Signal Strength Indication	20
2.6.5	Conclusion et choix de métrique	21
2.7	Conclusion	21
3	Conception	22
3.1	Scénario - Diagramme de séquence	22
3.1.1	Scénario A	22
3.1.2	Scénario B	22
3.1.3	Choix	23
3.2	Ensemble du système - Diagramme d'architecture	23
3.3	Schéma de Base de données	24
3.3.1	Tuple de données	24
3.3.2	Table par type de données	25
3.3.3	Stockage	25
3.4	Fréquence de relevé	26
3.5	Grafana	26
4	Expérimentations et résultats	31
4.1	Script de collection de données	31
4.1.1	Développements	31
4.1.2	Expérimentations	31
4.1.3	Résultats	32
4.2	Reproductibilité du protocole RPL	33
4.2.1	Développements	33
4.2.2	Expérimentations	34
4.2.3	Résultats	34
5	Planification	35
6	Fiches de suivi	39

7	Auto-contrôle et auto-évaluation	51
8	Bibliographie	53

Introduction

1.1 Sujet d'étude

Ce projet s'inscrit dans l'environnement FIT IoT Lab, une plateforme d'expérimentation pour l'internet des objets. Composée de 6 sites répartis dans toute la France, chacun d'entre eux contient un grand nombre de plusieurs types de nœuds, qui permettent différentes expériences à distance. Ces nœuds sont généralement des cartes électroniques, mobiles ou non, qui contiennent différents équipements :

- Capteurs
- Equipement radio pour communication avec des voisins
- LEDs
- Robot sur lequel est posée la carte

Il existe différents types de nœuds qui ont différents fonctionnements et buts, comme les capteurs m3 qui sont

les cartes que l'on utilise. Nous présenterons ces cartes plus en détail dans la suite du document.

Ces nœuds sont répartis à l'échelle d'un bâtiment dans chaque site de la plateforme, ce qui permet une cartographie par les cartes de ces sites.

1.2 Présentation de la problématique

En se basant sur cette plateforme, les chercheurs de l'Inria, à Lille, voudraient avoir à disposition des données temporelles sur leur bâtiment d'étude, qui pourrait dès lors s'apparenter à un bâtiment intelligent. Dès qu'elles seront disponibles, ces données constitueront une base libre qui, nous l'espérons, pourra être exploitée par les chercheurs pour différentes expérimentations. Ceci a donc pour but de faire gagner du temps à la recherche, ainsi qu'à fournir une méthode pour capter de nouvelles données. Ces données seront captées par des nœuds m3. Ces nœuds particuliers constituent une grande majorité du parc Lillois, et nous permettent donc d'obtenir une

masse de données importante sur ce site.

Par ailleurs, et grâce à cette masse de données, nous souhaitons étudier une seconde question : Dans une plateforme telle que FIT IoT, on peut se demander quel est l'impact des expériences des autres utilisateurs sur nos propres expérimentations. Nous souhaitons de fait étudier l'impact que peut avoir l'exploitation de la plateforme sur un protocole radio, et ainsi étudier la reproductibilité de ces expériences.

1.3 Objectifs poursuivis

Dans un premier temps, l'objectif principal est bien la mise en place de cette solution de captation automatique. En effet, ces capteurs étant répartis dans tout le bâtiment Lillois, il peut être intéressant de monitorer ces cartes afin d'obtenir des informations sur l'endroit, et ainsi potentiellement en faire un bâtiment intelligent.

Il faut donc que toutes les briques logicielles soient mises en place afin de pouvoir remplir une base de données à destination des chercheurs.

Par la suite, il nous faudra proposer une documentation extensive du travail fourni pour assurer une continuation du projet.

Puis une mise en place d'un protocole radio afin de mettre en évidence ou non, la possible perturbation radio induite entre expériences ou par l'environnement utilisé par de nombreuses personnes diverses (hall d'entrée d'un bâtiment).

1.4 Travail à réaliser et travail réalisé

Afin de satisfaire notre objectif principal ; la collecte de données automatiques de capteurs m3 ; il nous faut mettre en place plusieurs briques, qu'elles soient logicielles ou d'infrastructure. Dans notre problème, comme chaque partie de la plateforme est exploitée, il nous faut développer une solution pour chacune d'entre elles :

1. Au niveau des nœuds, il nous faut une solution logicielle de captation de données. Une fois captées, ces données doivent nous être renvoyées pour traitement et écriture dans la base de données.
2. Pour stocker les données captées par cette solution, il nous faut préparer et implémenter une base de données de type temporelle, qui est une solution adaptée à notre problématique liée au format des données : De très nombreuses petites mesures estampillées.
3. Enfin, il nous faut un applicatif qui se place entre ses deux solutions et qui permette de faire la jonction entre elles. Ici, ce serait un script qui contrôlerait tout, de la réservation des nœuds, à l'exploitation des données et leur envoi dans la base.

Notre second objectif, qui est donc d'observer l'impact des perturbations radio sur les expériences de la plateforme FIT IoT lab, nécessitera aussi plusieurs briques qui permettront de conclure

1. Il nous faut un protocole radio à étudier, afin de déterminer les effets de ces perturbations ;

2. Il nous faut surveiller ce protocole, récupérer plusieurs métriques quant à son fonctionnement, afin de déterminer si oui ou non il y a de réelles perturbations.
3. Enfin, il nous faut tester ce protocole dans des conditions volontairement perturbées afin de vérifier les limites de ce protocole.

Le premier objectif est totalement satisfait, à savoir que, la solution de collecte automatique est mise en place via plusieurs technologies :

1. Contiki pour le firmware ;
2. InfluxDB pour la base de données ;
3. Script python associé à cron pour l'automatisation ;

Le second objectif, en revanche, n'aura pas pu être mené à bien dans le temps imparti. En effet, nous avons défini cet objectif que relativement tard dans le projet, et n'y avons que peu réfléchi

De ce fait, bien que nous ayons mené une expérimentation, qui sera présentée dans la suite du rapport, la masse de données que nous avons récupérée est trop faible pour pouvoir établir une réelle conclusion. En effet, après une première partie où nous pensions avoir trouvé une réelle reproductibilité d'expérience, nous avons observé qu'en augmentant le nombre de nœuds sur lesquels nous effectuons nos expériences, et même dans le cas où la plateforme ne subit que très peu de perturbations, la reproductibilité du protocole n'est pas vérifiée. Il nous a de ce fait

été impossible de conclure quant à l'impact de l'utilisation de la plateforme.

Le but était de montrer que l'activité de la plateforme, que ce soit au travers des expériences ou de l'activité humaine sur le site de Lille, pouvait avoir un impact assez important sur les expériences menées par les utilisateurs, par exemple avec une expérience de routage. Nous voulions de ce fait utiliser les technologies suivantes :

1. Contiki pour le firmware des nœuds qui sont routés ;
2. Profil de monitoring pour vérifier le niveau de RSSI pendant l'expérience

1.5 Plan de l'étude

Ce rapport contient les différentes étapes du projet, à savoir la collecte de données automatique puis la captation radio et son impact réseau. Dans les parties à suivre, il y aura donc, premièrement, une présentation de la solution mêlée dans un second temps aux recherches sur les perturbations radio.

Le chapitre 2 se concentrera sur les études technologiques et la montée en compétence que nous avons effectuée sur ce début de projet. L'état de l'art sera donc technique avec la présentation de la plateforme, les firmwares M3 et les bases de données temporelles. Ensuite, une partie dédiée aux métriques réseaux et impacts sur la qualité des protocoles sera présentée.

Le chapitre 3 est dédié aux différents choix technologiques, diagrammes et concepts qui forment notre système, Ainsi qu'à l'exploitation du protocole réseau RPL

(Routing Protocol for Low-Power and Lossy Networks)
qui sera utilisé pour tester nos hypothèses. Il est central
car il constitue une base de travail solide sur laquelle se
retourner à tout moment lors du travail pour notre groupe.



État de l'art

2.1 Plateforme Fit IoT Lab

2.1.1 Présentation

IoT Lab fait partie du consortium FIT (Future Internet Testing facility) qui réunit plusieurs institutions de recherche et universités sur ce projet de plateforme de tests à grande échelle : FIT IoT Lab, FIT Wireless et FIT Cloud. FIT IoT Lab est une plateforme de service d'infrastructure IoT. Il existe sur cette plateforme plus de 1500 noeuds capteurs de différents types (M3,A8,turtlebot,...) répartis sur 6 sites : Grenoble, Lille, Lyon, Paris, Saclay et Strasbourg. L'objectif est de fournir aux chercheurs une infrastructure riche qu'ils peuvent ensuite réserver et exploiter pour leur propres expériences.

2.1.2 Architecture IoT Lab

La plateforme est décomposée en plusieurs parties (figure 2.3) :

- Site web - FIT IoT Lab (Webportal)

Il permet d'accéder au banc d'essai en ligne, réserver des noeuds capteurs pour une expérience avec un firmware dédié et une durée. Il contient aussi les tutoriels permettant de prendre en main la plateforme. Celui-ci est un client de l'API REST.

- L'API REST

Cette API REST contient une multitude d'appels pour différents besoins. On peut obtenir l'état du banc d'essai sur n'importe quel site ou alors se concentrer sur une expérience en cours ou passée pour obtenir des détails sur celle-ci. Elle contient par ailleurs des possibilités de modifier un firmware en cours d'expériences ou d'obtenir plus d'informations sur des robots en mobilité.

- Frontend SSH

La frontend SSH est un serveur distant auquel chaque utilisateur de FIT IoT Lab peut se connecter via son compte sur un site particulier, dans le

cadre de notre étude Lille. Elle est utile pour lancer directement des expériences, les gérer et prendre des informations diverses et variées. Cette frontend SSH s'appuie sur l'API REST pour fonctionner mais elle permet d'utiliser des scripts en local sur cet ordinateur distant afin de réaliser des tâches plus complexes comme le "serial aggregator" qui va agréger toutes les entrées des liens séries de nos noeuds. Les scripts sont écrits généralement en Python. De plus, cet accès permet par la suite de se connecter directement aux noeuds d'une expérience qu'on aurait lancé pour observer son comportement en détails.

- Noeuds capteurs

Il existe plusieurs types de noeuds : A8, M3, Firefly, Lora, Arduino, WSN430 et les turtlebots qui sont des unités mobiles. Chaque noeud à une documentation qui lui est propre. Cela permet une grande diversité dans l'expérimentation. Sur ces noeuds il faut ajouter un firmware pour obtenir le comportement par défaut attendu de chacun d'eux, le développement firmware s'effectue en C.

- Noeuds M3 (figure 2.1)

Il est essentiel de parler des capacités des noeuds M3 puisqu'ils seront notre objet d'étude sur les différentes problématiques traitées. La carte contient plusieurs capteurs de mesures ambiantes : lumière, pression, gyroscope, accéléromètre. Pour les communications radio, la carte utilise la norme 802.15.4 PHY 2.4 Ghz. La carte peut s'utiliser avec les dif-

férents OS (Operating System) suivants : Contiki, RIOT et FreeRTOS.

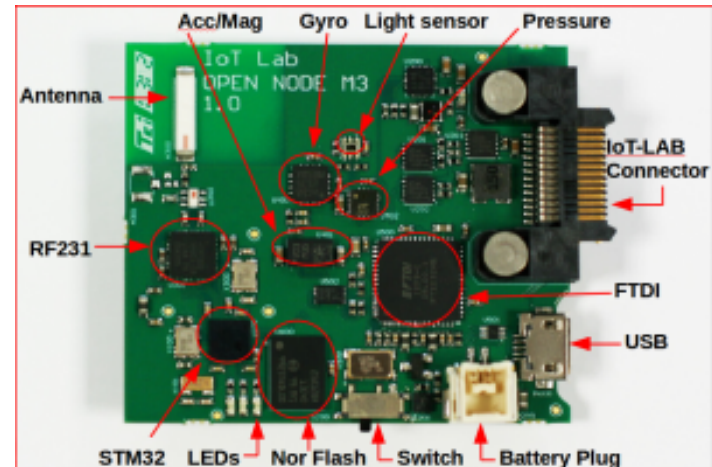


FIGURE 2.1 : Carte M3

Un noeud M3 est composé (figure 2.2) :

- De la carte M3 (Open Node) qui va être manipulée par l'utilisateur qui va avoir un accès total dessus (mémoire, OS et firmware utilisé).
- Une Gateway qui permet la connexion à cette carte et accès à son port série.
- Un noeud de contrôle (Control Node) qui va être le moyen d'envoyer ses instructions passives ou actives à la carte via la gateway et récupérer les données collectées de l'open-node.

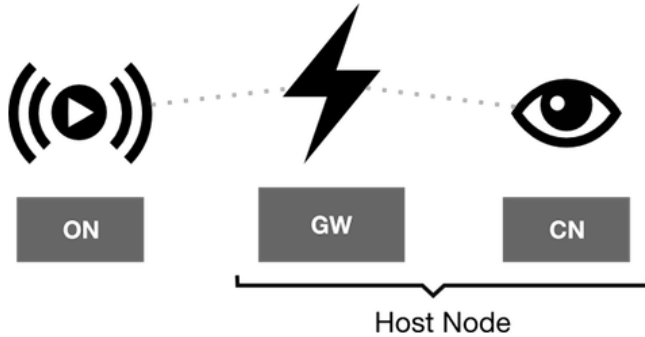


FIGURE 2.2 : Noeud M3

Dans la suite, nous discuterons indistinctement de la carte m3 et du noeud m3.

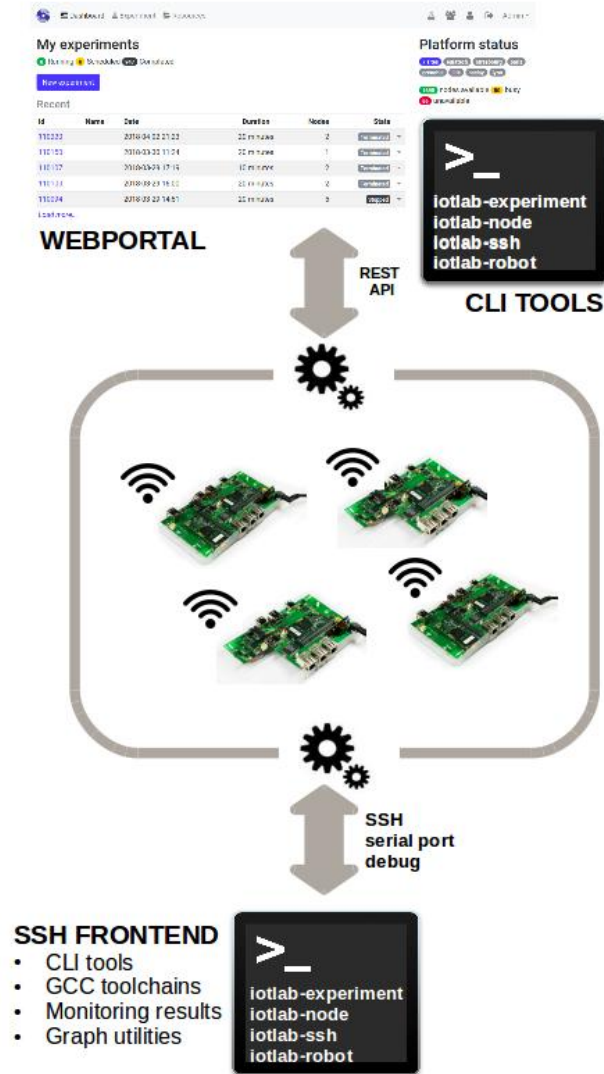
2.1.3 Tutoriels essentiels

Dans cette section, nous présenterons les tutoriels essentiels de la plateforme que nous avons dû suivre pour réaliser notre solution.

- Getting started : the tutorial for beginners [1]

Ce tutoriel vise à faire une première soumission d'expérience via la plateforme web : réserver des noeuds, ajouter un firmware et y accéder directement via SSH sur le banc d'essai. Cela nous a permis entre autre de découvrir la plateforme dans son ensemble et comprendre le but visé de celle-ci.

- SSH access [2]



Cette partie permet d'obtenir un accès SSH sur la frontend en faisant un échange de clé RSA via le site web. Nous avons donc pu grâce à ceci configurer un accès privilégié à notre compte sur la frontend SSH, d'ici nous avons pu déployer notre script et accéder aux noeuds d'une expérience en SSH.

- Experiment CLI Client [3]

Ce tutoriel fait partie d'un ensemble de tutoriels à propos des "CLI tools" qui permettent de faire des appels à l'API REST en soumettant des expériences, en flashant les firmwares des noeuds etc... Celui-ci permet d'apprendre la partie "experiment" qui concerne la soumission d'expérience ainsi que de son suivi. Cela nous permet donc d'apprendre la plus grande partie des commandes relatives aux CLI Tools et savoir sur quoi s'appuyer pour notre script comme la soumission d'expérience ou encore récupérer l'état de la plateforme afin de savoir la disponibilité de celle-ci.

- Nodes Serial Link Aggregation [4]

Une présentation du script "serial aggregator" qui permet d'agréger tous les liens séries relatifs à une expérience. Il permet aussi d'envoyer des messages à tous les liens qu'il agrège. Cette partie est essentielle pour notre projet puisque cet outil permettra d'agréger les données de tous nos noeuds M3 lors des expériences.

2.2 Firmwares

Comme expliqué par avant, la plateforme contient un grand nombre de nœuds capteurs qui permettent d'effectuer plusieurs expérimentations. Ces nœuds ont logiquement besoin d'une base logicielle pour fonctionner, et c'est ici qu'interviennent les firmwares. Il est central dans notre projet de programmer un firmware qui soit efficace et réponde à nos besoins. Pour cela, plusieurs solutions de systèmes d'exploitation sont disponibles, et présentées sur la plateforme FIT. Il faudra de fait effectuer un choix quant à la solution qui sera employée. Les différents systèmes étudiés peuvent être trouvés sur le site de FIT IoT ¹

2.2.1 Contiki

Contiki est un système d'exploitation open-source, désigné pour l'IoT, particulièrement pour la mise en place de protocoles réseaux (TCP/IP, IPv6, Low-Power protocols), reposant sur le système de proto-threads.

Avantages

Si nous nous sommes dans un premier temps engouffrés dans la solution Contiki, c'est que pour une première prise en main, la programmation semblait relativement simple. En effet, la plateforme FIT fournit du code exemple pour chaque système, et Contiki avait la base d'exemples la plus variée. De fait, l'aggrégation de deux

¹<https://www.iot-lab.info/operating-systems/>

exemples nous a permis d'arriver à une première solution plutôt acceptable.

Inconvénients

En y regardant de plus près, nous avons malheureusement remarqué que dans le cadre des nœuds m3, la plateforme Contiki n'a pas de librairie pour la lecture des capteurs de température, qui est une donnée que nous aurions aimé étudier. Par ailleurs, le temps de développement d'une telle librairie étant trop important pour ce genre de projet, nous avons préféré ne pas nous y pencher.

Ressources d'apprentissage

Le point de choix principal se trouve au niveau de la facilité d'apprentissage de la plateforme. De fait, la présence de nombreuses ressources est un point central dans le choix d'une technologie.

En l'occurrence, Contiki vient avec des tutoriels de base pour apprendre à compiler du code C en firmware Contiki². Ce qui le rend plus accessible, c'est surtout ses nombreux exemples qui permettent de rapidement comprendre comment sont programmés les Firmwares sur ce système d'exploitation.

Deux exemples, particulièrement, sont importants dans notre cas : Le premier, appelé 01-serial-echo, est un firmware qui, lorsqu'il reçoit un message sur son lien série, renvoie le message sur le lien série, à la manière de la commande echo. Ceci est important pour nous car nous

avons besoin de renvoyer les données sur le lien série, ainsi que recevoir par exemple une instruction de capture de données. Cette base de code contient aussi tout ce qui est nécessaire la base des mécanismes événementiels.

Le second, appelé 03-sensors-collecting, est un firmware qui, à chaque seconde, renvoie sur son lien série les valeurs captées par tous les capteurs définis au préalable. Ici, on obtient donc la base qui permet de capter des données sur les capteurs, ainsi que la manière de mettre un chronomètre en place, ce qui est important pour notre problème.

2.2.2 RIOT

RIOT est un système d'exploitation, qui vise les équipements ayant des ressources minimales, et à donc pour objectif l'efficacité en tous domaines : Énergie, mémoire, modularité... De même qu'avec Contiki, il est possible d'étudier de nombreux protocoles réseau grâce à ses librairies.

Avantages

RIOT vient tout comme Contiki avec une bonne base d'exemples et de tutoriels fournis par FIT IoT pour prendre facilement en main la plateforme. Par ailleurs, RIOT est plus régulièrement mis à jour et maintenu par sa communauté que Contiki.

²<https://www.iot-lab.info/tutorials/contiki-compilation/>

Inconvénients

De prime abord, le code permettant de programmer des firmwares grâce à RIOT semble relativement compliqué. C'est un système qui est plus obscur à comprendre de prime abord, et qui demande donc plus d'investissement pour être maîtrisé.

Ressources d'apprentissage

Pour la plateforme RIOT, les mêmes tutoriels existent que pour Contiki [5].

Par ailleurs, des exemples permettent aussi de comprendre le fonctionnement global de la plateforme et d'en extraire les principes de base. Malgré cela, il n'existe pas d'exemple traitant directement avec les capteurs, et il nous faut donc aller chercher plus en profondeur dans la documentation pour trouver des solutions à ce niveau.

2.2.3 Analyse

Étant donné le peu d'expertise que nous possédons dans le domaine, nous ne pouvons pas montrer définitivement qu'une solution est meilleure qu'une autre. En effet chacune d'entre elles nous permet à terme d'arriver à une solution très satisfaisante. En effet, le problème de base, au niveau du firmware, est relativement élémentaire et ne demande pas de programmation très compliquée.

Contiki, de ce fait, nous a semblé être une solution idéale pour avancer rapidement dans le problème et le résoudre efficacement.

2.3 Base de données temporelle

2.3.1 Présentation

Une fois la collecte de données automatisée, nous aurons des données horodatées mais il reste néanmoins la question d'où stocker ces données. Afin de mener le projet au bout, nous aurons besoin d'une base de données temporelle. En effet, celle-ci est indispensable pour stocker des données horodatées, non-relationnelles (mesures indépendantes les unes des autres), avec peu d'attributs et sur un grand volume.

2.3.2 Analyse

Tout d'abord, quelles sont les caractéristiques propres à une base de données temporelle ? Les Time Series Databases (notées TSDB) ont besoin d'un horodatage sur les données avec une proximité physique si l'horodatage est proche, les données sont à proximité afin de traiter celles-ci plus efficacement. Ensuite, elles se doivent d'exécuter des "range queries" rapidement, c'est à dire des requêtes rapportant toutes les données inférieures ou supérieures à une condition (ici la condition la plus utilisée sera le temps). Les TSDB se doivent d'obtenir une rapidité d'écriture élevée sur la base puisqu'elle sera remplie fréquemment, elles doivent donc rester le plus possible disponible afin d'éviter de perdre des données. En plus de ces aspects techniques, la TSDB choisie devra avoir une communauté active pour considérer qu'elle est fiable et qu'il y a de la documentation autour ou de l'aide communautaire pour maintenir la solution dans le temps. Pour

évaluer ces différents points, nous nous sommes référés à deux benchmarking fait en 2017 [6] et en 2019 [7].

Dans le premier benchmarking [6], il se concentre sur un aspect 'features' des bases de données, à savoir qu'elles caractéristiques ont-elles. Celui-ci pointe plusieurs aspects à savoir :

- Distribution/Clusterability : disponibilité, extensibilité, et un équilibreur de charge
- Functions : fonctions disponibles dans le système MIN/MAX/SUM
- Tags : contient un LTS (long time storage), possibilité d'avoir plusieurs horodatages sur une même valeur.
- Granularity : décrit le plus petit délai possible entre deux horodatages
- Interfaces and Extensibility : les APIs et plugins possibles
- Support and License : la propriété intellectuelle de la TSDB et son support commercial

Ce rapport montre qu'aucune base de données ne peut implémenter toutes les fonctionnalités existantes sur les TSDBs. Néanmoins, elle montre plusieurs avantages sur certaines TSDBs et leurs inconvénients.

Pour recentrer les recherches sur la solution à choisir nous avons donc pris en compte le classement de popularité des bases de données fait par db-engines (figure 2.4) [8].











Rank			DBMS	Database Model	Score	
Nov 2019	Oct 2019	Nov 2018			Nov 2019	Oct 2019
1.	1.	1.	InfluxDB 	Time Series	19.93	+0.31
2.	2.	2.	Kdb+ 	Time Series, Multi-model 	5.29	-0.15
3.	3.	 6.	Prometheus	Time Series	3.64	+0.04
4.	4.	 3.	Graphite	Time Series	3.32	-0.02
5.	5.	 4.	RRDtool	Time Series	2.90	+0.19
6.	6.	 5.	OpenTSDB	Time Series	2.13	+0.21
7.	7.	7.	Druid	Multi-model 	1.79	-0.05
8.	8.	8.	TimescaleDB 	Time Series, Multi-model 	1.73	+0.22

FIGURE 2.4 : Classement d'activité sur les bases de données temporelles

Celui-ci permet d'avoir une vision sur les bases qui sont très utilisées, mises à jour et donc actives.

Celles qui ressortent sont InfluxDB, KDB+ et Prometheus. InfluxDB étant loin devant les autres TSDBs.

- InfluxDB est une TSDB open-source qui a été conçue pour stocker et superviser ses données événementielles, en temps réel et est orientée IoT avec beaucoup de possibilités de visualisation.
- KDB+ est orienté aussi IoT et accentue son avantage sur la rapidité d'entrées / sorties de sa solution permettant de s'adapter aux besoins de grandes infrastructures industrielles.
- Prometheus est open-source et propose des outils de visualisation et des requêtes simplifiées afin d'accéder aux données et un système d'alertes avancé.

Afin d'affiner notre choix, nous avons pris en compte le deuxième benchmarking [7] qui se concentre sur des scénarios IoT et donc prend en compte la performance sur des requêtes SQL particulières, le volume de données utilisé pour une entrée ainsi que le nombre d'entrées possibles/sorties en même temps afin de savoir si la base de données à une bonne capacité d'écriture et de lecture. Néanmoins sur cette étude, on ne distingue pas d'exemple de données, il est discuté de données "générées" néanmoins dans un cadre IoT.

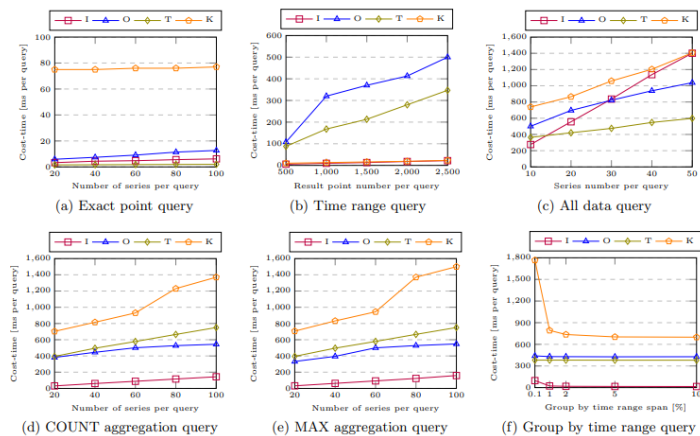


FIGURE 2.5 : Temps de réponse moyen en fonction de la charge [7]

Sur ces graphiques (figure 2.5), on observe le coût en temps de la requête en fonction du nombre d'entrées sur cette dernière. Lors de ce benchmarking, on peut observer qu'InfluxDB en rouge) [9] est nettement en avance sur les

autres TSDBs. Puisque le temps de réponse de celle-ci est bien inférieur aux différentes requêtes est bien inférieur. Là ou elle est moins performante est sur le "SELECT *" (All data query - c), or dans notre cas, les tables seront très courtes (une à deux données max par tables). C'est pourquoi notre premier choix technique se portera sur cette base de données. Néanmoins nous gardons à l'esprit d'autres possibilités de TSDBs, il sera donc important dans la partie conception de bien prendre en compte une certaine facilité à changer de TSDB si nécessaire. De plus, il n'est pas impossible par la suite de changer vers Prometheus [10] étant une TSDB déjà utilisée pour certains projets Fit IoT Lab.

2.4 Automatisation via Linux

Cette partie vise à déterminer quelles solutions existent sous Linux pour réaliser une tâche automatique comme l'exécution d'un script python qui est notre objectif

2.4.1 Cron

Cron ou crontab est un daemon linux qui dérive de "chrono table" en anglais. Celui-ci permet de lancer une tâche en arrière plan via un système simple. Pour exécuter une tâche crontab, il suffit d'éditer un fichier via la commande "crontab -e" et y mettre : un motif d'exécution horaire et notre commande. Par exemple, la ligne suivante : "0 */2 * * * python exemple.py" signifie qu'on exécute le script exemple.py toutes les 2 heures.

2.4.2 Systemd

Systemd est un système d'initialisation et un daemon linux. Il permet aussi d'automatiser les tâches sous la forme de services. Ainsi on peut lancer, arrêter, relancer le service si besoin. Pour la configuration, on utilise deux fichiers pour configurer respectivement : le temps auquel la tâche va s'exécuter et la tâche en elle même avec son environnement.

2.4.3 Comparaison

Systemd et cron sont deux solutions possibles pour notre script python. L'avantage de systemd est le fait qu'il soit plus complet et que plusieurs tâches peuvent être aisément exécutées et débuggées en même temps. Néanmoins, il introduit plus de complexité avec son système de fichiers, là où crontab ne nécessite qu'une ligne dans un script. Par ailleurs, crontab à un système de mail qui peut nous alerter si une tâche ne s'est pas effectuée correctement. Ainsi, étant donné que nous n'avons qu'une tâche à effectuer, il semble approprié de déployer crontab plutôt que systemd. Néanmoins, si par la suite, dans les évolutions futures de la solution, il était nécessaire d'avoir des scripts différents pour obtenir des relevés différents, nous recommanderions alors d'utiliser systemd.

2.5 Protocole RPL

RPL (Routing protocol for Low-power and Lossy networks)), est un protocole de routage qui est adapté à

l'IoT, défini dans la RFC 6550 [11]. Basé sur un système d'arbre de routage, ce protocole exploite deux types de nœuds : Dans le réseau, un client est désigné comme étant routeur de bordure. Ce routeur aura pour tâche de renvoyer les paquets remontant vers lui vers la branche qu'il faut. Les autres clients seront alors de simples clients qui renverront leurs paquets vers leur père dans la hiérarchie de l'arbre.

Nous nous basons sur une version basique de RPL, mais il existe des améliorations à ce protocole qui ont été proposées. Nous pouvons par exemple évoquer les améliorations apportées par M. Brandon FOUBERT, que l'on peut retrouver dans l'article suivant [12], dont il a présenté les principes et les résultats lors des journées RSD organisées à Nantes, auxquelles nous avons pu assister dans le cadre de la formation.

Dans notre optique d'analyser l'influence de l'activité humaine dans une plateforme telle que celle de FIT, nous avons besoin d'un protocole de routage léger qui soit facilement implémentable, étant donné le peu de temps qu'il nous restait pour travailler dessus. Par ailleurs, il devait être adapté aux réseaux mobiles, et avoir de potentielles problèmes.

RPL est un protocole qui remplit toutes ses caractéristiques. Étant déjà implémenté dans Contiki, sous FIT IoT lab , nous n'avons quasiment pas de temps de développement à prendre en compte. Par ailleurs, RPL est très sensible à des pertes de paquets, ce qui nous permet d'exploiter une solution qui puisse avoir des soucis en cas de bruit sur la plateforme.

2.6 Métriques réseau sur le bruit et qualité

Concernant la partie réseau, il nous fallait une métrique qui nous permettrait d'évaluer la qualité du réseau, ainsi que le bruit ambiant. Nous avons découvert trois métriques différentes, qui pourraient être acceptables dans notre sujet et qui permettent de faire ces deux choses. Il existe probablement d'autres indicateurs permettant de récupérer ces informations, mais nous avons préféré nous concentrer sur les trois suivantes :

- Le LQI : Link Quality Indicator
- L'ETX : Expected transmission count
- Le RSSI : Received Signal Strength Indication

Nous allons donc par la suite les présenter puis expliquer notre choix

2.6.1 R-ED - Reciever ED

Sur cette notion se base deux des prochaines métriques, et il nous faut donc succinctement la présenter ici. C'est une estimation de la puissance du signal reçu dans la bande de fréquence du canal en question. Cette estimation est indépendante des données reçues et ne doit jamais lire les données en question. C'est une notion définie dans la RFC 802.15.4 [13], et dont on peut retrouver une définition dans cette référence [14], à la page 402.

2.6.2 LQI : Link Quality Indicator

Le LQI est un indicateur de qualité de lien. En se basant sur le R-ED ou sur une estimation du rapport signal / bruit, l'implémentation décidée par les constructeurs renvoie une valeur entre 0 et 255. Cette valeur est alors une estimation de la qualité ou de la force du lien entre deux noeuds, pour un paquet donné. Une valeur de 0 correspond à un lien inutilisable, là où une valeur de 255 indique un lien parfait. Cette valeur est définie dans la RFC 802.15.4 [13] [14]

2.6.3 ETX : Expected transmission count

L'ETX est une métrique permettant de déterminer la qualité d'un chemin entre deux noeuds, dans un réseau sans-fil. Le calcul est relativement simple, c'est le nombre d'envois de paquets nécessaire pour recevoir un certain nombre de paquets sans aucune erreur. Par exemple, si l'on veut transmettre 1000 paquets, et qu'il nous faut 2000 transmissions pour tout transmettre sans erreur, alors la valeur de l'ETX est 2000/1000 soit 2. Ainsi un ETX de 1 signifie que l'on a un chemin qui ne pose jamais souci, et cette valeur peut monter jusqu'à l'infini, si par exemple le chemin est rompu. Cette valeur est définie dans l'article suivant [15]

2.6.4 RSSI : Received Signal Strength Indication

Le RSSI est une indication relative de la puissance reçue sur un canal, exprimée en dBm. Cette valeur permet de

calculer le bruit ambiant sur un noeud, et ainsi de déterminer l'activité radio ambiante à un moment donné. Cette valeur est relativement simple à récupérer dans FIT, en ajoutant un profil de monitoring à une expérience. Encore une fois ici, il n'existe pas de norme précisant la manière de calculer le RSSI, et l'implémentation est laissée libre au constructeur.

2.6.5 Conclusion et choix de métrique

Notre choix s'est de ce fait finalement porté sur le RSSI, et ce pour plusieurs raisons : L'implémentation et sa mise en place dans les expériences FIT IoT lab est évidente et rapide. Le besoin de cette métrique étant arrivé relativement tard dans le projet, nous n'aurions pas eu le temps de mettre en place une autre métrique. De ce fait, ce choix s'est naturellement fait directement. Malgré cela, c'est aussi la valeur qui correspond le plus à notre problématique, et qui nous permet le mieux de trouver une réponse. Bien que les autres métriques nous auraient aussi permis de conclure, nous pensons avoir fait le choix idéal avec cette celle-ci.

2.7 Conclusion

Sur cet état de l'art, nous avons présenté les différents éléments nécessaires à l'implémentation de notre solution à savoir :

- La plateforme FIT IoT Lab
- Les firmwares M3

- La base de données temporelles

Nous avons pu aboutir à une première solution en prenant des choix qui nous semblent corrects. A savoir le choix de développer sur la plateforme FIT via la frontend SSH afin de ne pas à avoir héberger notre script et directement lancer l'automatisation en local, mettre en place un premier firmware via Contiki puisqu'il offre une simplicité de programmation et déjà une bonne base d'exemple de firmwares et mettre en place dans un premier temps une base de données InfluxDB mais garder à l'esprit de bien compartimenter cette partie afin de ne pas avoir de problèmes d'intégration s'il y a une nécessité de migrer vers un autre type de base de données.

À l'issue de ce travail de recherche bibliographique, il apparaît que plusieurs propositions peuvent servir de base à la résolution de notre problème.

Il semble se détacher un certain nombre de directions privilégiées que nous allons exploiter en priorité dans nos propositions.

Celles-ci font l'objet de l'étude du chapitre suivant.



Conception

Cette partie est dédiée au système dans son ensemble et à la manière dont il a été pensé et conceptualisé. Nous voulons ici présenter les différents systèmes qui composent notre solution, ainsi que les différentes interactions qui les caractérisent. Les diagrammes ont pour but de clarifier le propos et de fluidifier la compréhension.

3.1 Scénario - Diagramme de séquence

Cette section a pour but d'expliquer le fonctionnement de notre script de collecte de données à la fin du projet pour que la partie majoritaire du projet soit réussie. Nous nous sommes penchés sur deux scénarios différents de diagramme de séquence.

3.1.1 Scénario A

Le scénario A (figure 3.4) s'exécute dans la séquence suivante :

1. Le script s'exécute automatiquement à une certaine heure de la journée, vérifie la disponibilité des nœuds et réserve un maximum d'entre eux
2. Une fois les nœuds réservés grâce à l'API REST, ils sont flashés avec le bon firmware et commencent leur capture de données
3. Puis on accède directement à un nœud pour faire une demande de données via une entrée prédéfinie. On ré-exécute ceci pour chaque nœud soit séquentiellement, soit via des threads
4. Une fois traitées, ces données sont formatées et envoyées vers la Base de données temporelles pour écriture et stockage

3.1.2 Scénario B

Le scénario B ressemble au scénario A mais il utilise le serial aggregator. Comme décrit dans la partie état de

l'art, le serial aggregator permet d'agréger toutes les données reçues par les noeuds M3 en un seul point. Le début et fin du scénario sont donc les mêmes. Ainsi, la différence décrite dans le diagramme en figure 3.5) est la suivante :

1. Initialisation du script comme précédemment avec flash des noeuds.
2. Dès lors, le Serial aggregator commence à écouter les nœuds et récupère les données qui lui sont envoyées par lien série. Une fois qu'il a tout reçu, il renvoie ces données au script Python qui les traite.
3. Récupération et mise en forme dans la base de données.

3.1.3 Choix

Le scénario A permet d'avoir un meilleur contrôle sur la captation puisqu'on va directement accéder aux noeuds néanmoins il nécessite une grande synchronisation et beaucoup de connexions en même temps à leurs interface. C'est pourquoi la deuxième approche avec le serial aggregator est préférée dans notre cas. Cela permet une implémentation plus aisée en utilisant les outils déjà développés mais aussi un seul point de sortie pour toutes les données. Le script python va donc être simplifié et plus lisible.

3.2 Ensemble du système - Diagramme d'architecture

Ce diagramme d'architecture (Figure 3.3) représente où sont placés les différentes parties prenantes pour l'exécution de la collecte automatique des données. Ainsi que d'un ordre partiel pour comprendre le mécanisme global. La solution nécessite 2 sites pour fonctionner : Grenoble là où l'API REST est localisé et Lille pour les noeuds M3. Sur Lille, trois composants sont nécessaires :

- Frontend SSH contenant un compte configuré avec le script DataCollector dessus et le crontab en arrière plan pour l'automatisation.
- Testbed de Lille, c'est à dire le bâtiment de Lille contenant tous les noeuds M3.
- Seveur Inria, un emplacement pour héberger la base de données.

Les différentes étapes de la solution sont :

- 1 - Exécution automatique, par le daemon Cron du script DataCollector. Crontab permet de paramétrer une ligne de commande à lancer toutes les X minutes où un schéma horaire particulier. On peut aussi utiliser systemd qui est un autre daemon réalisant la même tâche.
- 2 - Exécution de CLI Tools. Le script va faire une succession d'appels à l'API REST pour avoir plusieurs information. Premièrement, prendre l'état du

Testbed (combien de noeuds sont disponibles sur la plateforme), puis de lancer l'expérience en fonction avec notre firmware et un profil de monitoring radio. Enfin, attendre que l'expérience lancée soit prête pour démarrer l'étape 5.

- 3 - Programmation des noeuds. L'API REST va réserver les noeuds et donner le firmware associé à l'expérience.
- 4 - Flash des noeuds. On associe les noeuds à leur firmware, ils vont donc relever 5 données et les envoyer ensuite sur leurs liens séries.
- 5 - Lancement Serial Aggregator. Une fois l'expérience prête, on lance le script serial aggregator qui va écouter les liens séries de tous les noeuds d'une expérience.
- 6 - Ecoute des liens séries. L'expérience ne dure qu'une minute pour éviter de monopoliser la plateforme FIT de Lille. Au bout de la minute, les connections se ferment à tous les noeuds, ce qui arrête le serial aggregator qui va rassembler toutes les données.
- 7 - Ecriture dans la BDD. Le script DataCollector récupère la sortie du serial aggregator et prépare son format pour pousser ses données dans InfluxDB en JSON.

3.3 Schéma de Base de données

Nous avons eu deux propositions de schéma de base de données afin de savoir comment mieux représenter les données et leurs utilisations.

3.3.1 Tuple de données

La première idée est de faire un tuple de données avec : Noeud, Timestamp, LightValue, PressureValue, RssiValue, RssiChannel. Cela peut paraître à priori une bonne solution contenant toutes les informations d'un noeud à un instant T. Le premier problème est que les données ne sont pas prises au même moment exactement, ce qui signifie que les données RSSI, lumière et pression ne partagent pas la même estampille de temps. Sachant que les données seront traitées ensuite par des chercheurs pour entraîner de l'intelligence artificielle, il faut une précision dans les données. De plus, cette façon de conceptualisation revient à un schéma classique de base de données relationnelles. Or dans notre cas, nos données ne sont pas liées, excepté par leur numéro de noeud. Un autre problème qui pourrait arriver dans ce cas là, serait une absence de données sur un instant. Cela peut arriver pour plusieurs raisons : soit le noeud n'a pas été choisi dans l'expérience (ce qui est possible car lors de la soumission, on ne demande qu'un nombre de noeuds et non des noeuds précis) ou alors un problème sur le noeud directement, et ce pour diverses raisons. Nous savons que ceci peut exister puisque certains noeuds sont reconnus pour avoir des problèmes à l'initialisation.

3.3.2 Table par type de données

Une solution alternative peut être de représenter la base de données avec une table par type de donnée stockée. Cela permet de garder une estampille correcte avec la valeur de donnée associée. Afin de garder le lien entre les données, à savoir le noeud capté, on utilise un "tag" qui représente l'utilisateur de la ressource. Ainsi, toutes les données captés par un noeud M3-1XX seront marquées par ce même nom. On peut ainsi traiter les données sur ce tag. Le système InfluxQL favorise ce schéma puisqu'il est optimisé pour les requêtes InfluxQL (langage SQL pour InfluxDB). Le schéma de base de données temporelles final reste simple puisqu'il est composé seulement de 3 tables :

- Light, contient les données de lumière en lux
- Pressure, contient les données de pression en maBar
- RSSI, contient les données radio en dBm

3.3.3 Stockage

Comment InfluxDB stocke-t-il ses données ? Celui-ci découpe son organisation en 3 dossiers :

- Meta contient un certain nombre de paramètres et informations utiles : les users, databases existantes, retention policies, shards, and continuous queries. Les retentions policies sont un moyen de définir le recyclage de la base de données, c'est à dire à quelle moment va-t-elle supprimer des données ou archiver.

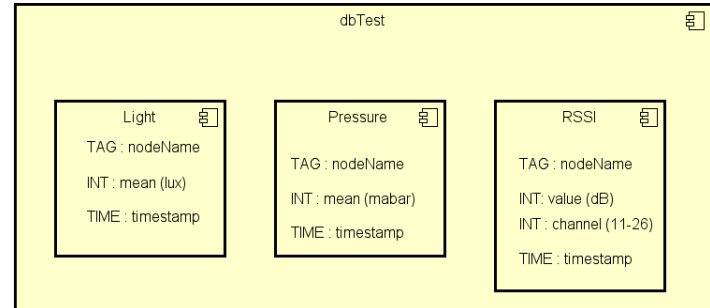


FIGURE 3.1 : Schéma base de données

Les shards sont la vision d'InfluxDB sur les données, elle groupe les données selon des shards (en général par proximité temporelle) et va les stocker, supprimer, traiter au même endroit. Les continuous queries sont un outil permettant de faire des requêtes en continue sur la base.

- WAL (Write Ahead Log) est un dossier de cache pour les bases de données.
- Data est le dossier contenant les bases de données. Il existe par défaut la base de données internal contenant un certain nombre d'informations générales sur nos bases de données. Puis ensuite, il existe toutes les données relatives à nos collectes.

3.4 Fréquence de relevé

Une fois la solution de collecte de données mise en place, il faut déterminer quand et comment lancer le script. L'exécution se fait par crontab mais combien de noeuds doit on réserver? A quelle fréquence? Pour le nombre de noeuds réservés, la solution réserve un taux de disponibilité de la plateforme. Par exemple à un instant T, la plateforme dispose de 300 noeuds dont 200 sont disponibles. Si le taux de réservation du script est 50% alors on réserve 100 noeuds sur Lille. Ainsi on évite de vouloir réserver plus de noeuds que disponible. Ensuite quel taux choisir? Idéalement 100% mais cela peut poser problème puisque la plateforme va délayer notre expérience car trop de noeuds seront indisponibles si elle accepte notre expérience. Ainsi nous avons décidé arbitrairement de prendre 70% de la plateforme sur une minute, ce qui limite qu'on affecte trop la disponibilité. Ensuite la fréquence s'effectue toutes les 30 minutes par soucis de disponibilité puisque si on intensifie ses relevés, trop de noeuds seraient indisponibles pour les autres utilisateurs. Mais aussi nous relevons des mesures physiques, or celles-ci ne vont pas avoir d'énormes variations entre plusieurs demi-heures. Par la suite, avec une vision sur plusieurs semaines, la fréquence de relevé pourra être ajusté pour diminuer la volumétrie ou l'augmenter selon les besoins des chercheurs.

3.5 Grafana

Après la collecte de données, il nous fallait un outil pour la visualisation des données et son exploitation. C'est pourquoi nous avons choisi Grafana qui est un outil web permettant de facilement intégrer les données InfluxDB via une interface web avec une vue simplifiée. Une vue classique de nos données peut se représenter ainsi : figure

[3.2](#)

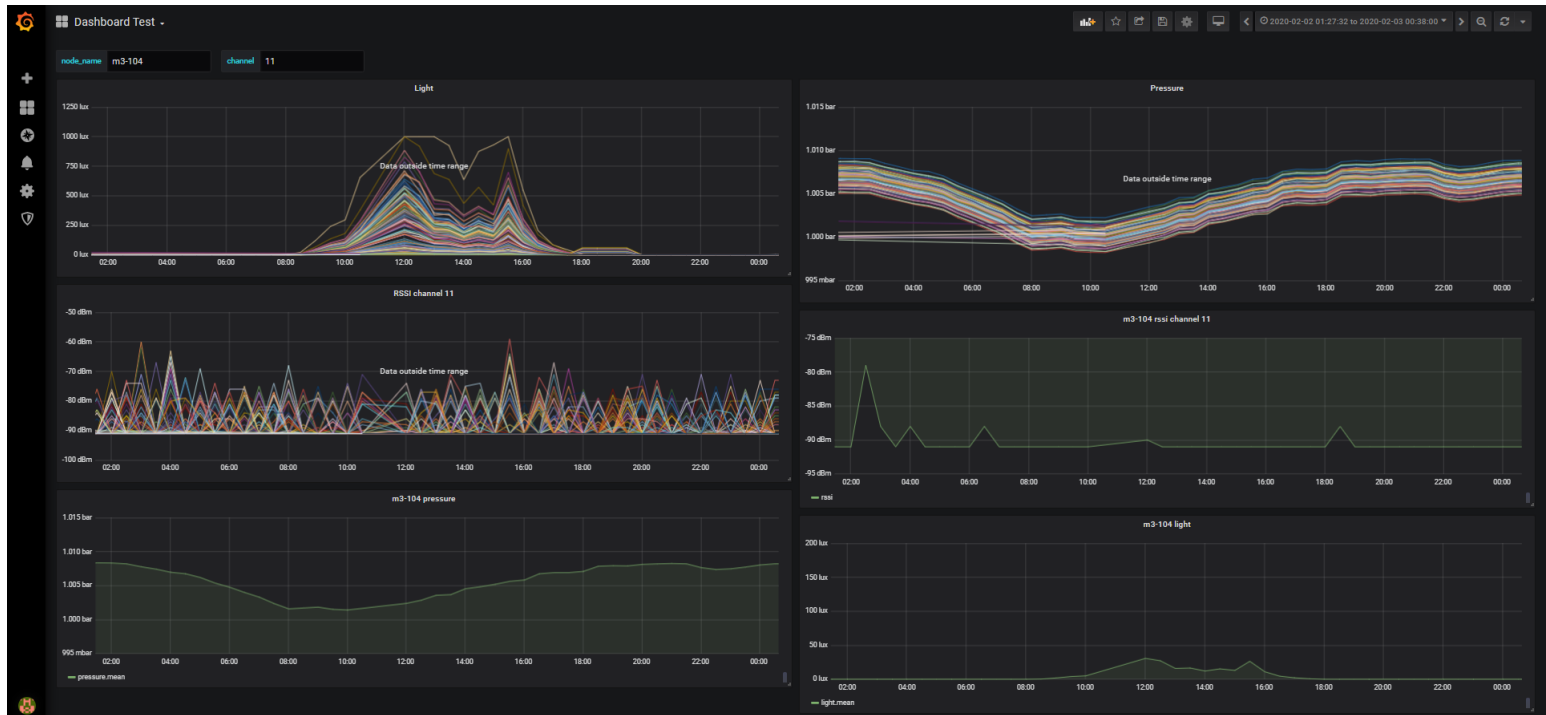


FIGURE 3.2 : Exemple de visualisation des données captées - 02/02/2020

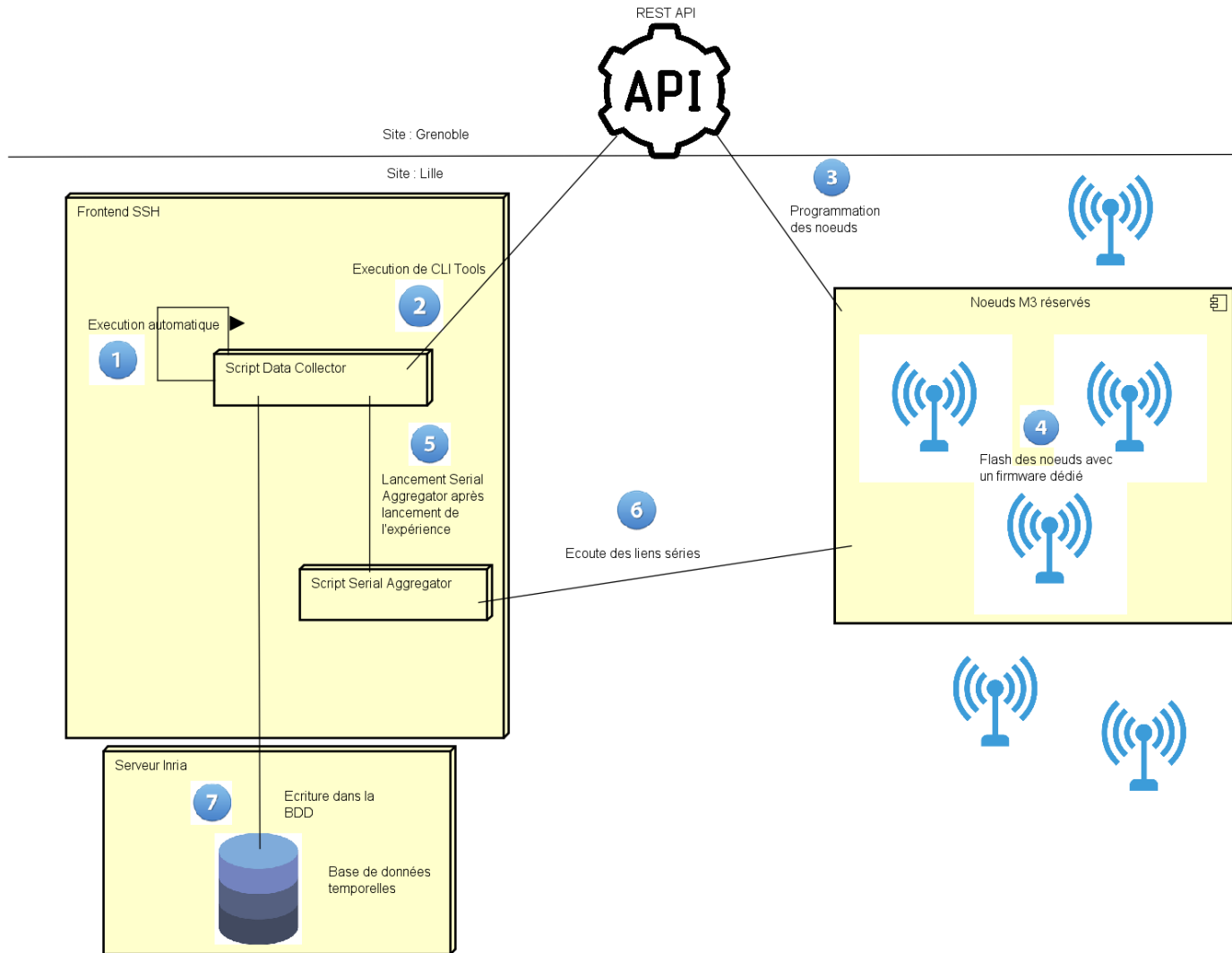


FIGURE 3.3 : Diagramme d'architecture

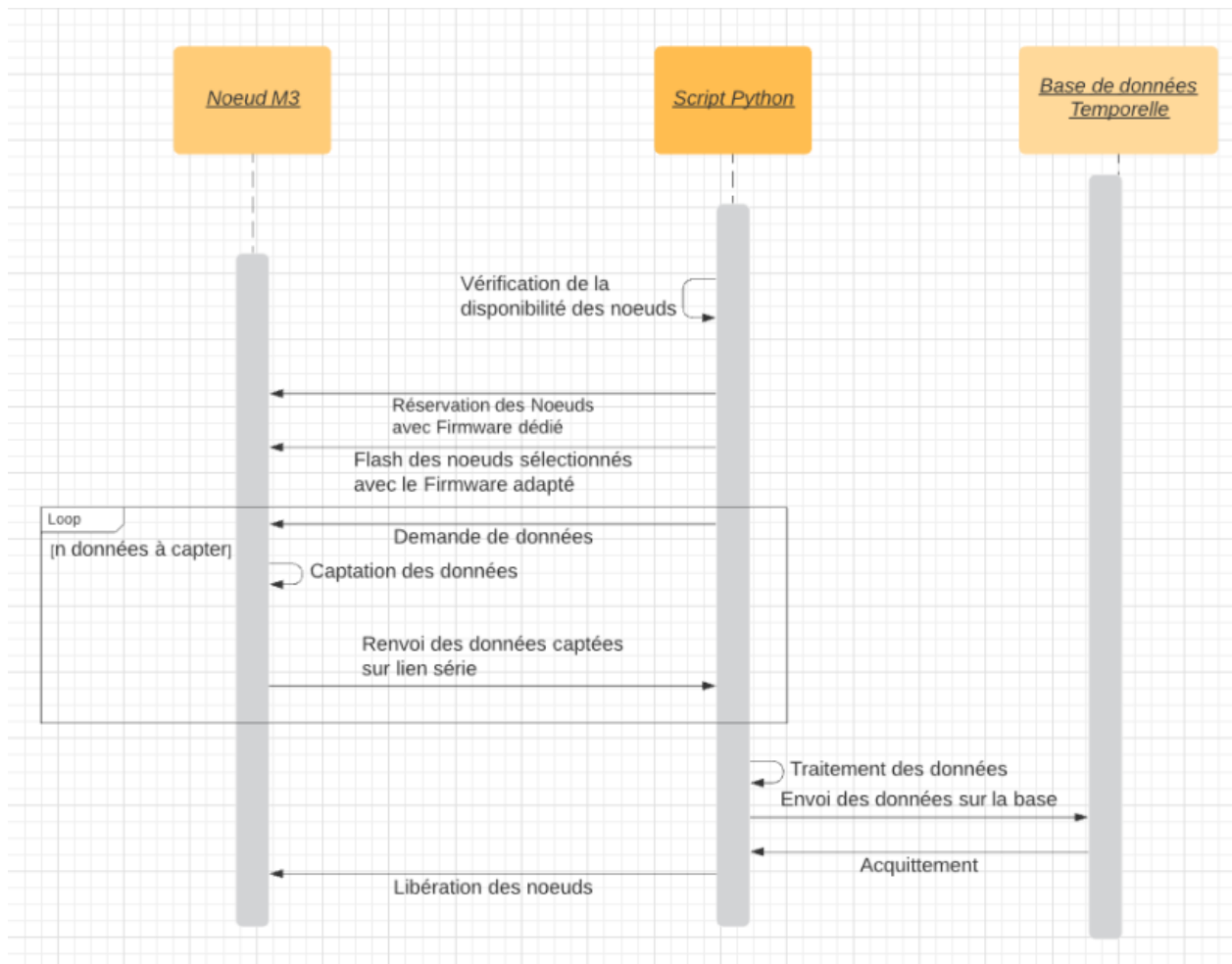


FIGURE 3.4 : Diagramme de séquence scénario A

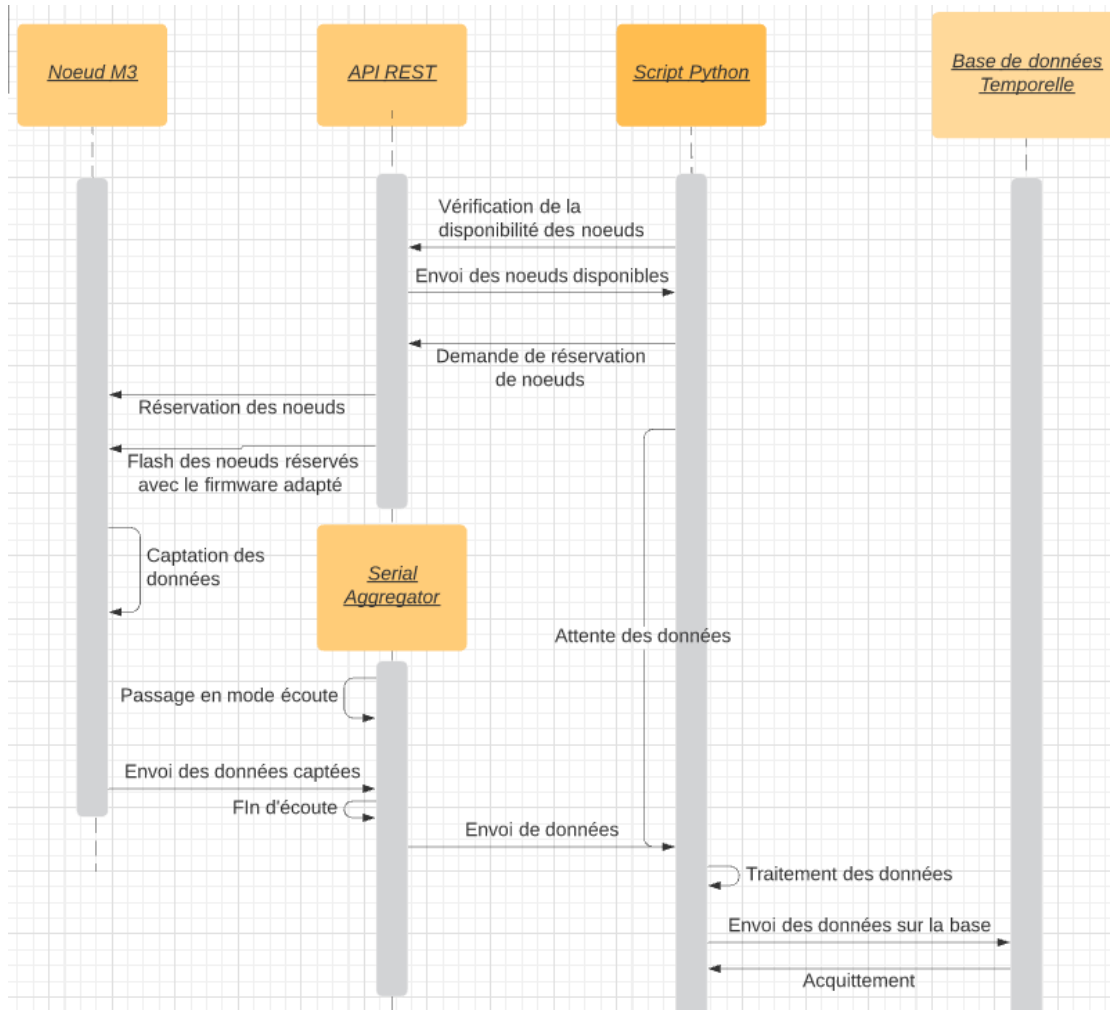


FIGURE 3.5 : Diagramme de séquence scénario B

Expérimentations et résultats

4.1 Script de collection de données

4.1.1 Développements

La solution de collecte de données (DataCollector) est totalement implémentée dans la plateforme FIT IoT Lab. Celle-ci respecte le schéma d'architecture 3.3 et diagramme de séquence 3.5.

4.1.2 Expérimentations

Nous avons fait plusieurs expériences pour mesurer le RSSI qui posait certains problèmes d'irrégularités 4.1. Dans un premier temps, nous nous sommes penchés sur comment le RSSI était pris dans nos mesures. Il s'avère qu'une mauvaise valeur était retournée mais il restait une chose étrange à savoir que les canaux étaient tous passifs (-91 dBm constant) excepté le premier. Dans cette démarche nous avons essayé de savoir si notre code était encore en défaut ou si l'utilisation d'un "monitor profile" (outil fourni par FIT pour capter les canaux radios) po-

sait un problème d'implémentation. Il s'avère que notre problème n'était pas dans ces 2 propositions mais venait de la fréquence de relevé d'expérimentation. En effet, la communication radio étant de l'ordre d'une dizaine de micro-secondes et notre période de 4s, il y avait un problème d'échelle. C'est pourquoi en intensifiant les relevés en dizaine de millisecondes, on retrouve des valeurs fluctuant sur tous les canaux.

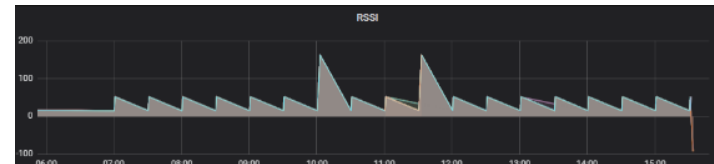


FIGURE 4.1 : Grafana - Bug sur le RSSI

4.1.3 Résultats

Via Grafana, on peut visualiser nos différentes données à savoir :

- Lumière : figure 4.2 . La lumière est mesurée en lux.

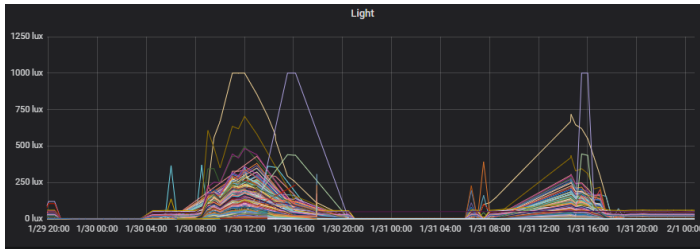


FIGURE 4.2 : Grafana - captation lumière 29-31/01/2020

Une première chose intéressante qu'on peut remarquer est le fait qu'on observe des pics de lumières vers 6H, ce qui équivaut aux agents d'entretiens du bâtiment puis que la lumière monte globalement au cours de la journée.

- Pression : figure 4.3 . La pression est en mBar. On peut observer que la pression atmosphérique est globalement la même pour tous les noeuds mais qu'il reste des variations. De plus on observe des lignes ne respectant pas la tendance globale qui sont dues à une absence de données de ses capteurs pendant le relevé, par exemple à cause d'une autre réservation de quelqu'un.
- RSSI : figure 4.5. Le RSSI est complexe à mesurer

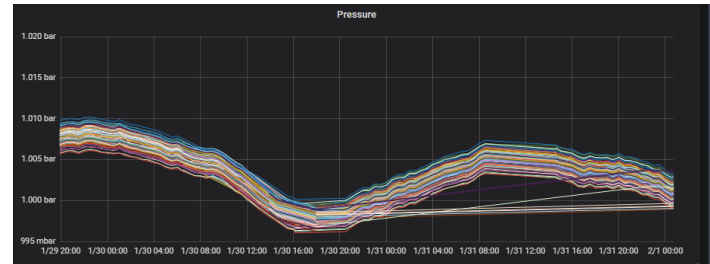


FIGURE 4.3 : Grafana - captation pression 29-31/01/2020

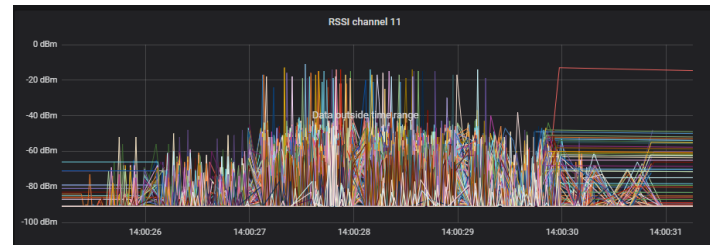


FIGURE 4.4 : Grafana - captation rssi 29-31/01/2020

puisque, comme dit précédemment, la fréquence de mesure influe sur ce qu'on va obtenir. Mais il est impossible de mettre une fréquence trop élevée comme 1 ms puisque cela augmenterait considérablement la taille des données. L'astuce utilisée est de mettre une fréquence assez faible (environ 10 ms) et de supprimer les fréquences à -91dbm lorsqu'il n'y a pas d'activité radio. C'est à dire que dès lors qu'il y a un changement de RSSI, celui ci va enregistrer son entrée mais aussi l'entrée précédente et future. Ainsi dès que deux valeurs consécutives de RSSI sont à -91dBm (équivalent à aucune activité radio particulière), on efface la donnée pour limiter un nombre très important de points non intéressants.

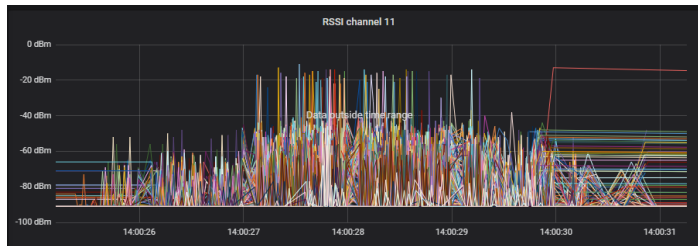


FIGURE 4.5 : Grafana - captation rssi channel 11
08/02/2020-14 :00 :26 à 14 :00 :32

Une vue par noeud 4.6 est mise aussi en place pour pouvoir mieux visualiser les valeurs de RSSI étant très rapprochées sur une petite période.

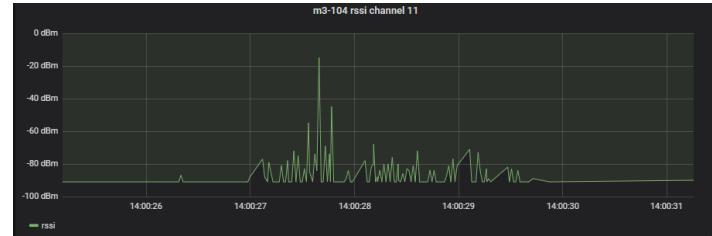


FIGURE 4.6 : Grafana - captation rssi channel 11, noeud
m3-104, 08/02/2020-14 :00 :26 à 14 :00 :32

4.2 Reproductibilité du protocole RPL

4.2.1 Développements

Cette partie est sûrement l'une de celles qui aura demandé le moins de développement. En effet, le protocole RPL possède déjà une implémentation dans FIT IoT lab, sous la technologie Contiki.

Par ailleurs, les différents scripts permettant d'affecter des préfixes IPv6 ou de regarder la topologie de l'arbre RPL créé sur chaque nœud est déjà fournie par la plateforme FIT. Il n'y a donc pas eu de développement quant à ces questions ci.

Le plus important a donc été la manière de penser cette expérience et de la designer afin qu'elle puisse nous apporter des solutions qui soient exploitables

4.2.2 Expérimentations

Nous utilisons donc le système d'expériences de FIT IoT Lab pour réserver 30 nœuds m3 sur la plateforme. Ces nœuds se doivent d'être assez proches afin que RPL puisse en router un maximum dans un arbre de routage. Dans ses 30 nœuds, on en sélectionne un (Si possible toujours le même), qui sera alors le routeur de bordure, c'est à dire la racine de l'arbre défini par RPL. Les autres nœuds sont de simples clients qui sont chargés de générer du trafic en temps normal (Ici, ils ne font que se router)

Dès que l'expérience est démarrée, nous affectons un préfixe IPv6 aux nœuds que nous avons réservés grâce à la frontale SSH. Sur cette même frontale, nous vérifions alors le bon établissement de l'arbre de routage RPL. Dès que la solution a convergé, on note le temps que l'expérience a pris et on recommence.

La difficulté vient avant tout du fait que cette expérience n'est qu'assez peu reproductible. En effet, à chaque fois que l'on recommence, la topologie de l'arbre est différente, certains nœuds ne rentrent pas dans l'arbre etc..

4.2.3 Résultats

Malheureusement, et étant donné le court temps que nous avons eu pour effectuer ces expériences, les données récoltées ne nous ont pas permis de conclure. En effet, si pour un petit nombre de nœuds, RPL se comporte très bien, ce n'est pas le cas pour un nombre plus grand de cartes. Plusieurs essais à 30 nœuds, à des périodes durant lesquelles la plateforme Lilloise n'était pas utilisée,

RPL n'a jamais réussi à connecter les 30 nœuds dans un seul arbre. Le maximum de connexions que nous avons obtenu était 14, ce qui est trop faible pour estimer cette expérience comme une réussite. Toutes les autres tentatives se sont soldées par des échecs avec des arbres d'une taille de 8 à 10 nœuds.

Avec plus de temps nous aurions pu nous pencher plus en avant sur les raisons de ces échecs, qui existent forcément puisque plusieurs personnes ont déjà utilisé RPL sur FIT IoT lab sans que cela pose de soucis.

Planification

Cette partie inclut des graphiques relatifs à notre travail. Nous avons découpés nos tâches en 5 catégories :

- Tutoriel FIT IoT Lab
- Recherches bibliographiques
- Développement Firmware/Python
- Rédaction de rapport, bilan de réunions,...
- Contact : les réunions et appels

Dans l'ensemble, l'estimation de nos tâches ont été proches de la réalité mais nous avons globalement sous-estimé les heures nécessaires pour réaliser notre travail. Nous avons donc dû augmenter nos horaires sur certaines semaines plus libres que d'autres. Cela nous a permis de nous reconcentrer sur des petites tâches comme conseillé par Julien Vandaele pour obtenir une démo pour la première période (7 octobre 2019 - 01 décembre 2019).

La deuxième période présente plus de développement afin de finaliser la solution de collecte automatique. Puis quelques recherches bibliographiques et expérimentations pour compléter notre maîtrise du protocole RPL. Nous avons eu un gros contre temps avec l'automatisation du script qui présentait pas mal de difficultés d'implémentation au sein de la plateforme FIT.

Temps effectif et estimé par catégorie - Période 1

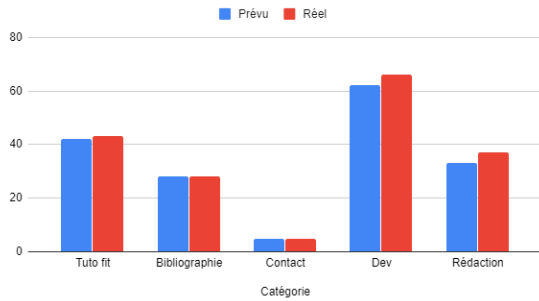


FIGURE 5.1 : Temps par catégorie de tâche - période 1

Répartition des tâches - Période 1

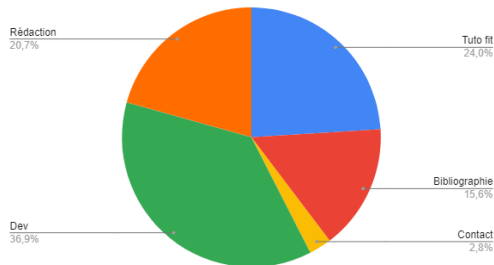


FIGURE 5.2 : Répartition du temps par catégorie - période 1

Temps effectif et estimé par catégorie - Période 2

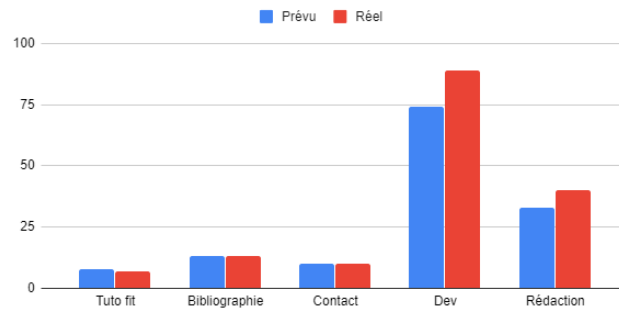


FIGURE 5.3 : Temps par catégorie de tâche - période 2

Répartition des tâches par catégorie - Période 2

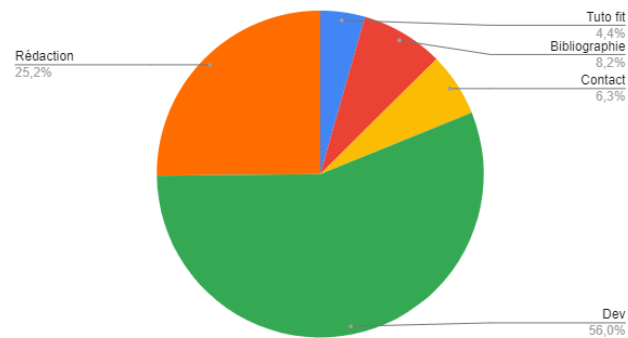


FIGURE 5.4 : Répartition du temps par catégorie - période 2

Tâches période 1

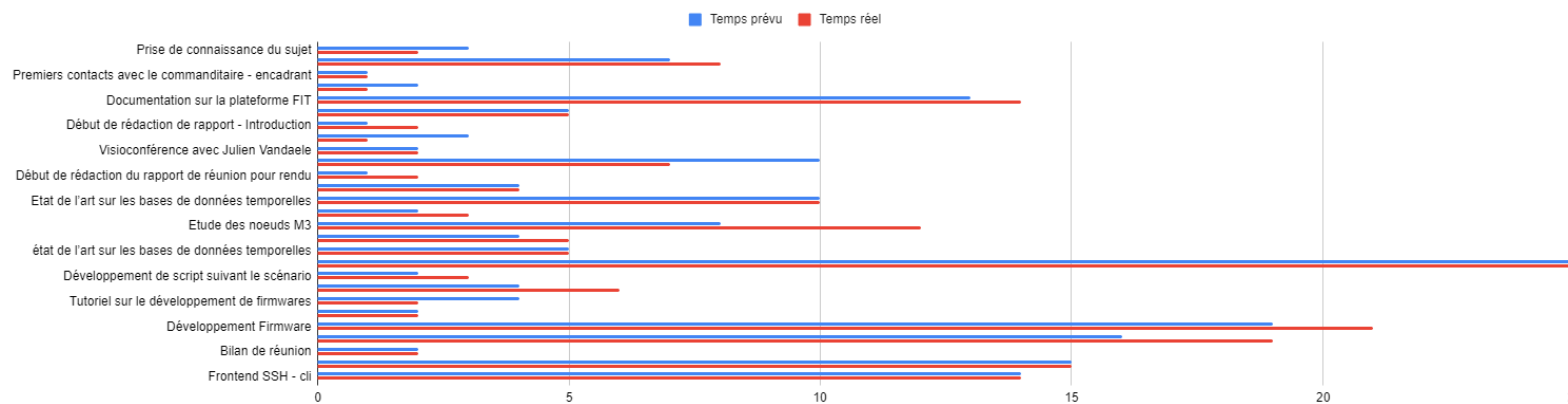


FIGURE 5.5 : Planning effectif - période 1

Tâches période 2

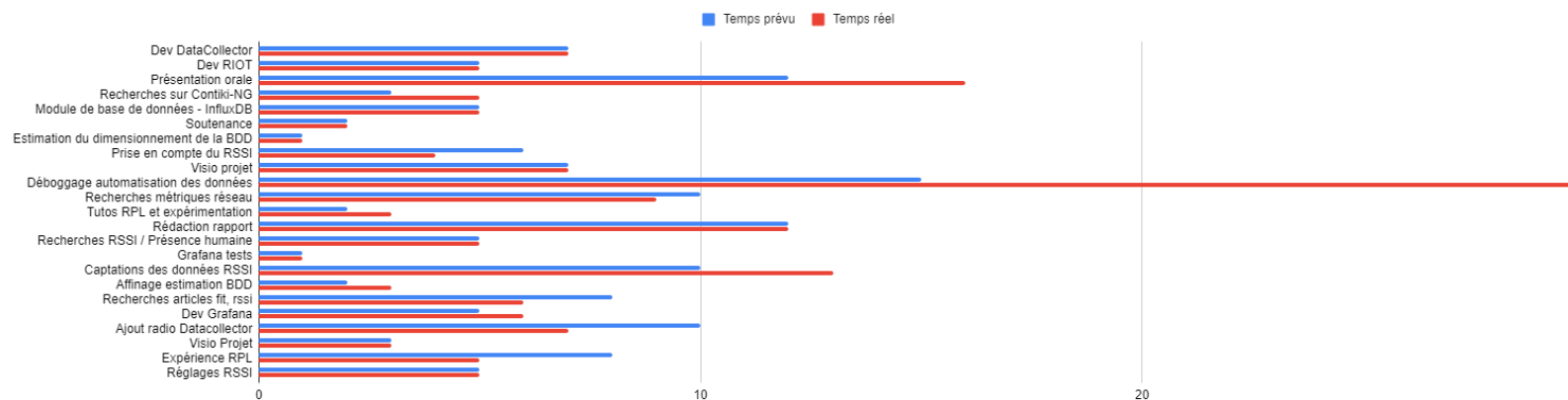


FIGURE 5.6 : Planning effectif - période 2

Fiches de suivi

Cette annexe est *obligatoire*.

Fiche de suivi de la semaine 1 du 7 Octobre 2019 au 13 Octobre 2019

Temps de travail de Pierre-Adrien DELISLE: 6 h 30 m

Temps de travail de Samuele DA SILVA: 5 h 30 m

Travail effectué.

- Prise de connaissance du sujet
- Prise en main de la plateforme FIT IoT lab
- Premiers contacts avec le commanditaire - encadrant
- Début de recherches bibliographiques

Travail non effectué.

Échanges avec le commanditaire.

- Demande d'informations auprès de Julien Vandaele
 - Précisions du sujet et questions générales

- Début de reformulation du sujet

Planification pour la semaine prochaine.

- Travail plus profond sur le sujet - Recherche de problématique
- Recherches bibliographiques
- Prise en main de la plateforme FIT IoT lab

Fiche de suivi de la semaine 2 du 14 Octobre 2019 au 20 Octobre 2019

Temps de travail de Pierre-Adrien DELISLE: 11 h 00 m

Temps de travail de Samuele DA SILVA: 11 h 00 m

Travail effectué.

- Documentation sur la plateforme FIT
 - Nodes M3
 - Tutoriels FIT IoT lab
 - API REST FIT IoT lab
- Début de tests de code sur l'API REST de FIT IoT lab
- Début de rédaction de rapport - Introduction
- Prise en main d'InfluxDB

Travail non effectué.

Échanges avec le commanditaire.

- Mails de présentation et précisions du sujet par mail

Planification pour la semaine prochaine.

- Continuation des recherches bibliographiques
- Tutoriels FIT IoT lab
- Rédaction de l'introduction du rapport
- Visioconférence

Temps de travail de Pierre-Adrien DELISLE: 5 h 30 m

Temps de travail de Samuele DA SILVA: 5 h 30 m

Travail effectué.

- Visioconférence avec Julien Vandaele, qui aura permis de finalement délimiter le sujet et lui donner un cadre précis
- Tutoriels FIT IoT lab : SSH / Build firmwares
- Début de rédaction du rapport de réunion pour rendu la semaine prochaine

Travail non effectué.

- Plus de tutoriels
- Rédaction du rapport
- Lectures bibliographiques
- Tout ce travail n'ayant pas été effectué à cause d'une semaine chargée

Échanges avec le commanditaire.

- Visioconférence

Planification pour la semaine prochaine.

- Lectures bibliographiques / Étude de la plateforme FIT IoT
 - Impact de la présence humaine sur divers protocoles

- Tutoriels FIT IoT lab
- Début de programmation d'un firmware de test

Fiche de suivi de la semaine 4 du 28 Octobre 2019 au 3 Novembre 2019

Temps de travail de Pierre-Adrien DELISLE: 3 h 30 m

Temps de travail de Samuele DA SILVA: 3 h 30 m

Travail effectué.

- Début de listing des possibilités pour les Systèmes de bases de données temporelles

Travail non effectué.

- Travail bibliographique
- Etude de la plateforme FIT IoT lab

Échanges avec le commanditaire.

Planification pour la semaine prochaine.

- Rattrapage du travail non effectué lors des deux dernières semaines
 - Recherches bibliographiques

- Etude de la plateforme FIT IoT lab

- Listing et choix d'une base de données idéale
- Début de travail sur un premier firmware pour expériences sur capteurs

Fiche de suivi de la semaine 5 du 4 Novembre 2019 au 10 Novembre 2019

Temps de travail de Pierre-Adrien DELISLE: 15 h 00 m

Temps de travail de Samuele DA SILVA: 15 h 00 m

Travail effectué.

- Etat de l'art sur les bases de données temporelles
 - Préparation d'une présentation de solutions
- Diagrammes de séquence pour s'assurer de la marche à suivre pour le développement
- Etude des noeuds M3
- Début de développement d'une démonstration simple de script de collecte de bases de données

Travail non effectué.

Échanges avec le commanditaire.

- Échanges autour de la séquence d'expérimentation, précisions finales du sujet

Planification pour la semaine prochaine.

- Réunion prévue le 12/11 à 15h
- Rédaction du rapport
- Développement du script suivant le scénario
- Recherches bibliographiques centrées sur l'interaction Humains/réseaux wireless

Fiche de suivi de la semaine 6 du 11 Novembre 2019 au 17 Novembre 2019

Temps de travail de Pierre-Adrien DELISLE: 10 h 00 m

Temps de travail de Samuele DA SILVA: 9 h 00 m

Travail effectué.

- Suite de l'état de l'art sur les bases de données temporelles
- Rédaction du rapport
 - Rédaction de l'état de l'art sur les bases de données

- Présentation de la plateforme FIT IoT lab

- Développement de script suivant le scénario

- Diagrammes de séquence validés par le commanditaire et les encadrants

- Recherches bibliographiques centrées sur l'interaction humains/réseaux wireless

- Ces recherches n'ont pas été pertinentes puisqu'aucun article en lien avec notre problématique n'était dans le sujet. Par ailleurs, cette question ne sera traitée que bien plus tard, il n'était donc pas nécessaire d'investir du temps dès maintenant sur le sujet

- Tutoriel sur le développement de firmwares

- Tutoriel Contiki
- Tutoriel de départ

Travail non effectué.

Échanges avec le commanditaire.

- Réunion le 12/11 (1h30)

- Présentation de l'état de l'art sur les BDD temporelles
- Reformulation et recentrage sur les besoins du projet
- Projet abordé de façon plus agile que normalement

- Attendus pour la prochaine réunion (Présentation Powerpoint pour la soutenance, Rapport, étape firmware)

Planification pour la semaine prochaine.

- Poursuite de la découverte des tutoriels firmware en réalisation d'une première réalisation de firmware (développement C
- Etat de l'art sur les différentes plateformes de développement des firmware
- Apprentissage d'une librairie Python en vue d'une utilisation de SSH pour se connecter sur FIT IoT lab

- Analyse du code des firmware existants
- Installation de l'environnement de travail en local
- Début de programmation d'un firmware maison

• Python SSH

- Recherche de différents packages existants pour utiliser SSH en python (Paramiko - Fabric - Quelles différences ? - Que choisir ?)
- tutoriels sur les packages en question
- Premier test de connexion SSH sur FIT via Python
- Scénario basique de l'algorithme

Travail non effectué.

Échanges avec le commanditaire.

- Bilan de réunion du 12/11

Planification pour la semaine prochaine.

• Python SSH

- Ecrire la liste des commandes SSH à réaliser
- Commencer à intégrer le scénario dans Python

- Finition du firmware maison objectif pour la réunion de la semaine suivante

Fiche de suivi de la semaine 7 du 17 Novembre 2019 au 24 Novembre 2019

Temps de travail de Pierre-Adrien DELISLE: 17 h 00 m

Temps de travail de Samuele DA SILVA: 19 h 00 m

Travail effectué.

- Développement Firmware
 - Recherche dans les firmwares exemple existant afin de déterminer si certains sont liés à notre problème

Fiche de suivi de la semaine 8
du 24 Novembre 2019 au 01 Décembre 2019

Temps de travail de Pierre-Adrien DELISLE: 28 h 0 m

Temps de travail de Samuele DA SILVA: 25 h 0 m

Travail effectué.

- Développement Firmware
 - Développement, test et debug du firmware Sur Contiki
 - Intégration au code de Pierre-adrien
 - Début de tests sur RIOT
- Python SSH
 - Dev Fabric Python
 - => Méthode python SSH avortée (voir échange avec le commanditaire)
- Frontend SSH
 - Librairie subprocess / os
 - Dev subprocess => exécution de commande bash
 - Format des données
- Rédaction du rapport
 - Introduction
 - Etat de l'art : mise en forme

- Conception
- Autres

Travail non effectué.

Échanges avec le commanditaire.

- Discussion Python SSH

Julien a fait une remarque sur Python SSH ne voyant pas l'utilité de celui-ci. Notre point de vue était de réaliser le script python en local puis se connecter en SSH par la suite sur le frontend SSH afin de lancer les commandes pour lancer l'expérience, récolter les coordonnées,... La vision de Julien est de mettre le script directement sur le frontend SSH pour ne pas avoir à l'héberger par la suite. Ainsi aucune connexion SSH serait nécessaire. => Le développement sur le frontend SSH a abouti en utilisant la librairie os, subprocess pour lancer les commandes
- Librairie manquante pour l'exploitation du capteur de température des noeuds m3 en contiki
 - Librairie manquante pour l'exploitation du capteur de
 - Possibilité que cela existe en contiki-ng
 - Besoin potentiel de créer ce code
 - Ou de changer de solution (Riot par exemple)

Planification pour la semaine prochaine.

- Développement Démo

- Ajouter de la généricité pour le script
- Automatiser le script
- Firmware RIOT
- Préparation soutenance

- Présentation des slides et pré-soutenance

Planification pour la semaine prochaine.

- Finalisation de la présentation
- Continuer le développement de l'automatisation
- Fin des tests et du développement sur RIOT

Fiche de suivi de la semaine 9 du 01 Décembre 2019 au 08 Décembre 2019

Temps de travail de Pierre-Adrien DELISLE: 11 h 30 m

Temps de travail de Samuele DA SILVA: 8 h 0 m

Travail effectué.

- Développement de la démonstration pour la soutenance
- Développement avec Crontab pour automatiser le lancement des scripts afin de collecter automatiquement les données
- Suite du développement du Firmware avec RIOT
- Préparation de la présentation

Travail non effectué.

Échanges avec le commanditaire.

Fiche de suivi de la semaine 10 du 8 Décembre 2019 au 15 Décembre 2019

Temps de travail de Pierre-Adrien DELISLE: 11 h 0 m

Temps de travail de Samuele DA SILVA: 11 h 0 m

Travail effectué.

- Présentation de la présentation pour la soutenance
- Recherches dans contiki-ng
- Modules de BDD
- Soutenance intermédiaire

Travail non effectué.

Échanges avec le commanditaire.

- Questions à propos de l'hébergement de la BDD

Planification pour la semaine prochaine.

- Redéfinir un sujet sur quant à l'aspect recherche du projet
- Continuer l'intégration à la base avec l'hébergement à trouver
- Rechercher dans RIOT la librairie traitant du capteur température puisque absent dans Contiki-NG

Fiche de suivi de la semaine 11 du 15 Décembre 2019 au 22 Décembre 2019

Temps de travail de Pierre-Adrien DELISLE: 5 h 0 m

Temps de travail de Samuele DA SILVA: 5 h 0 m

Travail effectué.

- Première estimation du dimensionnement de la base de données
- Prise en compte du RSSI dans la solution de captation automatique de données

Travail non effectué.

Échanges avec le commanditaire.

- Visio-conférence à propos du projet

Planification pour la semaine prochaine.

- Estimer à nouveau le dimensionnement de la base de données de façon plus fine
- Recherches sur le RSSI et début d'utilisation

Fiche de suivi de la semaine 12 du 23 Décembre 2019 au 29 Décembre 2019

Temps de travail de Pierre-Adrien DELISLE: 0 h 0 m

Temps de travail de Samuele DA SILVA: 0 h 0 m

Aucun travail effectué car vacances de Noël

Fiche de suivi de la semaine 13 du 30 Décembre 2019 au 5 Janvier 2020

Temps de travail de Pierre-Adrien DELISLE: 0 h 0 m

Temps de travail de Samuele DA SILVA: 0 h 0 m

Aucun travail effectué car vacances de Noël

Fiche de suivi de la semaine 14
du 6 Janvier 2020 au 12 Janvier 2020

Temps de travail de Pierre-Adrien DELISLE: 10 h 30 m

Temps de travail de Samuele DA SILVA: 10 h 0 m

Travail effectué.

- Recherches bibliographiques sur le lien entre RSSI / Présence humaine
- Tests de captation des données RSSI sur les nœuds M3
- Affinage de l'estimation de stockage des données dans une base InfluxDB
- Captation du RSSI pendant la captation de données
- Premiers tests d'affichage des données avec Grafana

Travail non effectué.

Échanges avec le commanditaire.

Planification pour la semaine prochaine.

- Poursuivre les recherches bibliographiques
- Amorcer la rédaction du rapport final

- Finaliser le code de captation du RSSI
- Design d'une expérience afin de répondre à notre problématique

Fiche de suivi de la semaine 15
du 13 Janvier 2020 au 19 Janvier 2020

Temps de travail de Pierre-Adrien DELISLE: 11 h 30 m

Temps de travail de Samuele DA SILVA: 9 h 0 m

Travail effectué.

- Lecture et recherche d'articles
- Travail sur Grafana
- Inclusion du RSSI dans les données capturées
- Ajustement du schéma de la base de données

Travail non effectué.

Échanges avec le commanditaire.

- Réunion visioconférence pour définir les objectifs finaux du projet
- Echanges mail autour d'un souci sur CRON

Planification pour la semaine prochaine.

- Réunion visio-conférence
- Finalisation du dimensionnement de la base de données
- Développement des expériences de collecte de données
- Début d'exploitation des données collectées
- Finalisation de la recherche bibliographiques
- Début de rédaction du rapport final

Fiche de suivi de la semaine 16 du 20 Janvier 2020 au 26 Janvier 2020

Temps de travail de Pierre-Adrien DELISLE: 25 h 0 m

Temps de travail de Samuele DA SILVA: 23 h 0 m

Travail effectué.

- Débogage de l'automatisation de la collecte de données
- Recherches bibliographiques

- Rédaction rapport

Travail non effectué.

Échanges avec le commanditaire.

- Réunion visio-conférence

Planification pour la semaine prochaine.

- Collecte de données et exploitation
- Rédaction du rapport
- Réunion visio-conférence

Fiche de suivi de la semaine 17 du 27 Janvier 2020 au 2 Février 2020

Temps de travail de Pierre-Adrien DELISLE: 17 h 0 m

Temps de travail de Samuele DA SILVA: 20 h 0 m

Travail effectué.

- Expérimentations RPL avancés
- Travail sur le rapport final
- Réunion visio-conférence

- Ajustement fréquences RSSI

Travail non effectué.

Échanges avec le commanditaire.

Planification pour la semaine prochaine.

**Fiche de suivi de la semaine 18
du 3 Février 2020 au 9 Février 2020**

Temps de travail de Pierre-Adrien DELISLE: 23 h 0 m

Temps de travail de Samuele DA SILVA: 25 h 0 m

Travail effectué.

- Finalisation du rapport
- Ajout Script : drop des valeurs RSSI inutiles
- Exploitation RPL

Travail non effectué.

Échanges avec le commanditaire.

Planification pour la semaine prochaine.

dèle de fiche de suivi fourni, il vous faudra l'établir vous-même. Dans le cas contraire, une commande permet de le générer automatiquement avec le texte qui le référence et des hyper-liens vers chacune des fiches (paragraphe ci-dessous).

Le tableau 6.1 récapitule le taux d'avancement du projet. Rappelons que le temps de travail théorique *minimal* correspond au temps indiqué sur la maquette pédagogique auquel on ajoute un strict minimum de 20 % correspondant au travail personnel hors emploi du temps. La partie « haute » de la fourchette correspond à 50 % de temps supplémentaire au titre du travail personnel.

Le tableau récapitulatif du temps consacré au projet est *obligatoire*. Si vous n'utilisez pas strictement le mo-

Semaine	Temps prévu		Pierre-Adrien DELISLE			Samuele DA SILVA		
	bas	haut	hebdo.	Σ	%	hebdo.	Σ	%
	h : m	h : m	h : m	h : m		h : m	h : m	
1	10 : 00	12 : 30	6 : 30	6 : 30	65 (52)	5 : 30	5 : 30	55 (44)
2	20 : 00	25 : 00	11 : 00	17 : 30	87 (70)	11 : 00	16 : 30	82 (66)
3	30 : 00	37 : 30	5 : 30	23 : 00	76 (61)	5 : 30	22 : 00	73 (58)
4	40 : 00	50 : 00	3 : 30	26 : 30	66 (53)	3 : 30	25 : 30	63 (51)
5	50 : 00	62 : 30	15 : 00	41 : 30	83 (66)	15 : 00	40 : 30	81 (64)
6	60 : 00	75 : 00	10 : 00	51 : 30	85 (68)	9 : 00	49 : 30	82 (66)
7	70 : 00	87 : 30	17 : 00	68 : 30	97 (78)	19 : 00	68 : 30	97 (78)
8	80 : 00	100 : 00	28 : 0	96 : 30	120 (96)	25 : 0	93 : 30	116 (93)
9	90 : 00	112 : 30	11 : 30	108 : 00	120 (96)	8 : 0	101 : 30	112 (90)
10	100 : 00	125 : 00	11 : 0	119 : 00	119 (95)	11 : 0	112 : 30	112 (90)
11	110 : 00	137 : 30	5 : 0	124 : 00	112 (90)	5 : 0	117 : 30	106 (85)
12	120 : 00	150 : 00	0 : 0	124 : 00	103 (82)	0 : 0	117 : 30	97 (78)
13	130 : 00	162 : 30	0 : 0	124 : 00	95 (76)	0 : 0	117 : 30	90 (72)
14	140 : 00	175 : 00	10 : 30	134 : 30	96 (76)	10 : 0	127 : 30	91 (72)
15	150 : 00	187 : 30	11 : 30	146 : 00	97 (77)	9 : 0	136 : 30	91 (72)
16	160 : 00	200 : 00	25 : 0	171 : 00	106 (85)	23 : 0	159 : 30	99 (79)
17	170 : 00	212 : 30	17 : 0	188 : 00	110 (88)	20 : 0	179 : 30	105 (84)
18	180 : 00	225 : 00	23 : 0	211 : 00	117 (93)	25 : 0	204 : 30	113 (90)

TABLE 6.1 : Avancement du projet par rapport au temps de travail théorique minimal (respectivement haut)



Auto-contrôle et auto-évaluation



Bibliographie

- [1] Getting started with IoT-LAB : the tutorial for beginners - <https://www.iot-lab.info/tutorials/getting-started-tutorial/>
- [2] Configure your SSH access - <https://www.iot-lab.info/tutorials/ssh-access/>
- [3] Experiment CLI client - <https://www.iot-lab.info/tutorials/iotlab-experimenttools-client/>
- [4] Nodes Serial Link Aggregation - <https://www.iot-lab.info/tutorials/serial-aggregator/>
- [5] - Compilation with RIOT, <https://www.iot-lab.info/tutorials/riot-compilation/>
- [6] Survey and Comparison of Open Source Time Series Databases - Andreas Bader, Oliver Kopp, Michael Falkenthal - 2017
- [7] Benchmarking Time Series Databases with IoTDB-Benchmark for IoT Scenarios - Rui Liu, Jun Yuan - 2019
- [8] DB-Engines Ranking, <https://db-engines.com/en/ranking/time+series+dbms>
- [9] Time Series Database (TSDB) Explained | InfluxDB - <https://www.influxdata.com/time-series-database/>
- [10] Prometheus - Monitoring system and time series database - <https://prometheus.io/>
- [11] RPL : IPv6 Routing Protocol for Low-Power and Lossy Networks <https://tools.ietf.org/html/rfc6550>
- [12] Sharing is caring : a cooperation scheme for RPL network resilience and efficiency - 2019 - Brandon Foubert, Julien Montavont - hal-02095410
- [13] IEEE Standard for Low-Rate Wireless Networks

- IEEE 2015 - https://standards.ieee.org/content/ieee-standards/en/standard/802_15_4-2015.html

[14] IEEE Standard for Low-Rate Wireless Networks IEEE 2015
https://www.silabs.com/content/usergenerated/asi/cloud/attachments/siliconlabs/en/community/wireless/proprietary/forum/jcr:content/content/primary/qna/802_15_4_promiscuous-tbzR/hivukadin_vukadi-iTXQ/802.15.4-2015.pdf

[15] High-Throughput Routing for Multi-Hop Wireless Networks - Douglas S.J. De Couto - MIT 2004 - <https://pdos.lcs.mit.edu/papers/grid:decouto-phd/thesis.pdf>