Numéro d'ordre : **Université Lille 1 - Sciences et Technologies** **Année 2018**

THESE

présentée et soutenue publiquement le 19 septembre 2018

En vue de l'obtention du grade de

DOCTEUR DE L'UNIVERSITÉ LILLE I

Discipline : Informatique

par

Nadir Cherifi

Assistance au développement de logiciels embarqués contraints en énergie

Composition du jury

Rapporteurs :	Samia Bouzefrane	Maître de conférence HDR - CEDRIC, CNAM, Paris
	Jean-Philippe Diguet	Directeur de recherche - LAB-STICC, CNRS
Invité :	Colombe Herault	Docteur RnD - Worldline
Directeur de thèse :	Gilles Grimaud	Professeur - CRIStAL, IRCICA, Université Lille 1
Co-encadrants de thèse :	Thomas Vantroys	Maître de conférence - CRIStAL, IRCICA, Université Lille 1
	Alexandre Boé	Maître de conférence - IEMN, IRCICA, Université Lille 1

Centre de Recherche en Informatique, Signal et Automatique de Lille - Équipe 2XS

Remerciements

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Je courrais toujours pour aller partout,
mais je ne pensais pas pour autant que ça allait me mener quelque part.
Tom Hanks, Forest Gump (1994)

1.1 Contexte de la thèse

Ce qu'écrivait Mark Weiser en 1991 [Wei91], *"The most profound technologies are those that disappear. They weaves themselves into the fabric of everyday life until they are indistinguisable from it."*, devient depuis peu à peu réalité avec la profusion grandissante d'objets connectés répondant à des usages ou des besoins extrêmement variés.

Dans le cadre de notre travail, nous allons nous focaliser sur les aspects énergétiques et plus particulièrement sur le soutien des activités du développeur afin que l'énergie soit prise en compte dès la conception du logiciel embarqué sur l'objet.

L'Internet des Objets

Aujourd'hui, de plus en plus de petits objets physiques intelligents et autonomes sont interconnectés et fonctionnent autour de nous, afin d'accomplir différents types d'applications, allant des maisons intelligentes à l'informatique "vestimentaire" en passant par la surveillance médicale des patients ou encore surveillance de structures civiles.

Le terme d'Internet des Objets (ang. *Internet of Things, IoT*) regroupe un ensemble vaste de systèmes différents. Il est cependant possible de trouver les invariants de ces objets comme le montre la figure 1.1.

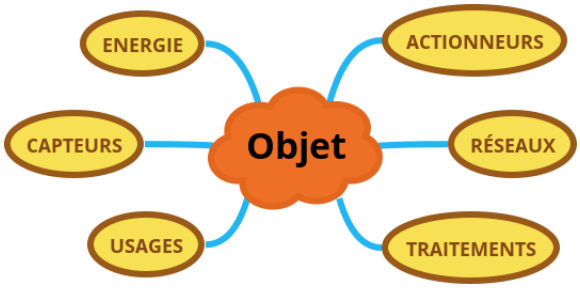


FIGURE 1.1 – Éléments constitutifs d'un objet connecté.

L'Internet des objets représente l'idée d'un système où des objets physiques communicants sont connectés directement ou indirectement à l'Internet "traditionnel" via des connexions filaires ou sans-fil.

Chaque objet connecté répond à un usage particulier. Il se compose d'un ensemble de capteurs permettant de récupérer des informations en lien avec ses différentes tâches. Ces informations peuvent être traitées localement et/ou à distance, et provoquer la mise en action d'un ensemble d'actionneurs. Le dernier point réside dans les aspects énergétiques (stockage, utilisation et récupération) car une grande partie de ces objets ne disposent pas d'une alimentation continue. Ils vont fonctionner grâce à des batteries.

Le "M2M" ou "*Machine To Machine*" fait partie intégrante de cette idée. On peut le voir comme l'implémentation de cette tendance, dans le contexte de l'industrie avec un fort axe à la remontée de données des objets vers des services informatiques distants.

Cette définition très large des objets que nous utilisons regroupe donc aussi bien les petits objets comme ceux que nous retrouvons dans le cadre des réseaux de capteurs, mais peut aller jusqu'au *smartphones*. Dans nos travaux de thèse, nous nous concentrons sur des petits objets qui sont pour le plus souvent basés sur des microcontrôleurs (i.e. nettement moins puissants que ce qu'on retrouve dans des *smartphones*). Ces systèmes représentent le cœur des objets que nous retrouvons dans l'Internet des Objets d'aujourd'hui.

Consommation énergétiques des objets connectés

Au vu de l'essor actuel de l'Internet des Objets avec un nombre d'objets dépassant le nombre d'humains sur la planète, des grandes prévisions sont faites par rapport au nombre d'objets connectés déployés dans un futur proche (20 Milliards d'objets aux horizon 2020 selon le cabinet Gartner [Gar17]).

De par la diversification des secteurs d'application, une intelligence toujours plus complexe doit être embarquée sur l'objet connecté. Embarquer sur des petits objets des applications de plus en plus complexes représente un vrai défi pour le développeur. Durant la conception, le développeur doit faire face à des contraintes de puissance et de mémoire disponibles sur le système, mais aussi à des contraintes énergétiques dictées par l'activité de l'objet et la capacité de la batterie.

La consommation énergétique d'un objet connecté autonome sur batterie est ce qui définit sa durée de vie. Ce phénomène physique représente donc une valeur critique, sachant que le remplacement d'une batterie peut être lourd, difficile et même parfois impossible (cas des objets/capteurs enfouis dans le béton, par exemple). Outre

cette contrainte importante sur la durée de vie après déploiement d'un objet, la consommation énergétique de celui-ci possède un impact bien plus global. En effet, selon l'Agence internationale de l'énergie (IEA) [Age14], la consommation énergétique dû à un milliard d'objets déployés dans les années à venir aura une contribution non négligeable dans la consommation énergétique globale mondiale. La contrainte de consommation énergétique prend donc aussi du sens pour des objets qui sont alimentés en continu via une source inépuisable.

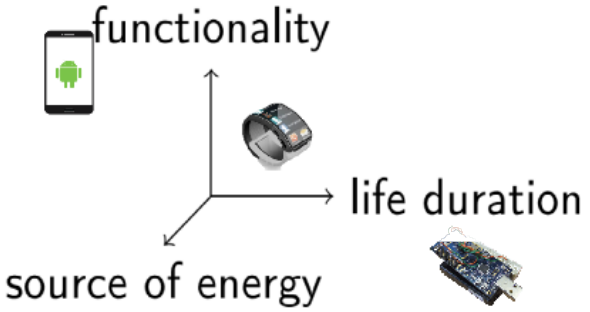


FIGURE 1.2 – Axes d'utilisation de l'énergie.

Ainsi, nous pouvons constater que la consommation énergétique représente un critère d'évaluation de première importance pour un logiciel embarqué à l'instar de l'empreinte mémoire ou encore des cycles processeur. Durant la conception d'une application embarquée, un développeur se retrouve donc souvent à jongler avec ces trois contraintes. Sur un plan énergétique, l'objectif premier est de trouver un compromis entre les fonctionnalités de l'objet prévues dans le cahier des charges, sa durée de vie ainsi que la capacité des sources d'énergie disponibles (Figure 1.2).

Contexte industriel de la thèse

La thèse décrite dans ce manuscrit s'écrit dans un contexte fortement industriel. Les travaux sont effectués en étroite collaboration avec l'entreprise Worldline dans le cadre d'une thèse CIFRE.

La société Worldline opère déjà un certain nombre de services utilisant des objets connectés. Le projet de compteur électrique intelligent « Linky » pour ERDF s'inscrit dans le domaine du *smart metering* et il représente le projet le plus important à l'heure actuelle. D'autres projets d'envergure ont démarré depuis quatre ans tels que la gestion des services pour les voitures connectées Renault, la télémétrie via des modules embarqués sur des camions équipés en seconde main par un grand équipementier ou encore de l'électroménager (four, réfrigérateur) connecté pour la société BOSCH/Siemens. Récemment, un projet d'objet connecté a été initié et développé. Celui-ci nommée *"Connected Kitchen"* permet une préparation assistée et facilitée de la liste de courses d'un foyer.

De nombreux clients de la société Worldline sont en train d'évaluer l'impact que peuvent avoir les objets connectés sur leur marché. Par exemple, dans le domaine de la santé avec le suivi de patient à domicile, du transport avec la multiplication des outils de suivi en temps réel, des média et de la grande distribution via l'utilisation de la *"Connected Kitchen"* pour diversifier les canaux de communication.

Tous ces éléments indiquent l'évolution constante de Worldline dans un monde où les objets connectés occupent une place prépondérante. Il y a donc une nécessité du point de vue industriel de monter en compétence sur ce secteur et de surmonter les défis encore existants. Dans le cadre de nos travaux, nous allons nous intéresser au défi de la consommation énergétique au sein de ces objets.

1.2 Objectifs de la thèse

L'énergie est une ressource critique dans le monde des objets connectés dont la consommation doit être optimisée. Le développeur d'application est en charge de détecter les bugs énergétiques pouvant être présents dans le logiciel afin de concevoir l'application embarquée la plus efficace en énergie possible. Notre but dans ces travaux de thèse est de venir en aide et en support au développeur d'application dans le but d'une meilleure compréhension de la contrainte énergétique lors du cycle développement.

Atteindre ce but passe par plusieurs étapes. Le critère le plus important pour nous est de placer la consommation énergétique au centre du processus de développement du logiciel embarqué. Ce processus se base donc sur ce critère énergétique afin d'améliorer le logiciel à chaque itération. Nous illustrons en Figure 1.3, une vue générale de ce processus, cible de nos travaux de thèse.

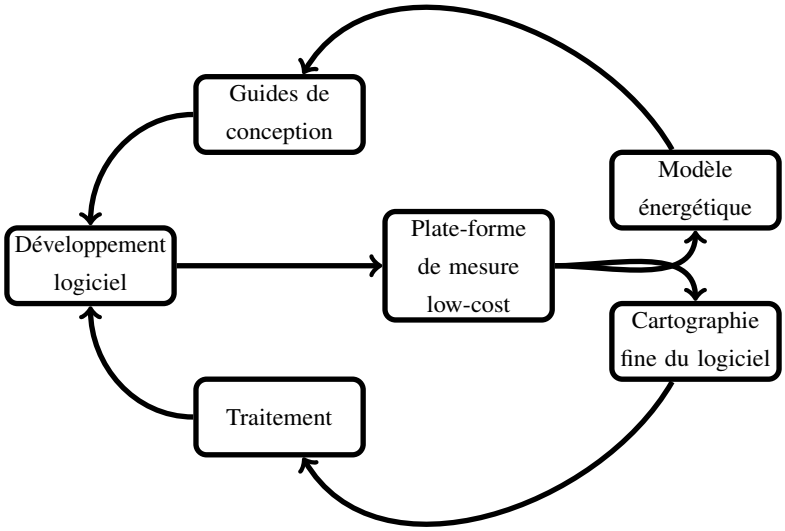


FIGURE 1.3 – Processus de développement.

Dans un premier temps, résumé dans la partie inférieur de la figure 1.3, nous avons l'objectif de faciliter au développeur la tâche, souvent ardue, de la mesure énergétique d'une application embarquée. La mesure effective du logiciel s'effectuera via une plate-forme bas-coût afin de fournir au développeur des données de cartographie

énergétique du code source de l'application. Ces étapes décrivent le premier cycle d'itération de notre processus de développement.

La deuxième partie de notre processus, décrit dans la partie inférieur supérieur de la figure, a pour but d'être proactif dans l'aide apportée au développeur. En effet, celle-ci, via l'analyse de profils de consommation énergétique, apporte au développeur des guides de bonnes pratiques. L'application et le suivi de ces guides permettent d'atteindre une efficacité énergétique adéquate pour l'application embarquée développée sur l'objet connecté.

1.3 Structure du document

Ce document de thèse s'articule autours de sept chapitres, incluant ce chapitre d'introduction.

Le **chapitre 2** présente un état de l'art consacré aux approches de profilage de la consommation énergétique d'une application embarquée. Il décrit aussi le bagages technique nécessaire pour l'appréhension et la compréhension du phénomène de dissipation énergétiques ainsi que sa mesure.

Le **chapitre 3** présente une expérimentation de l'état de l'art consacré aux approches d'estimation de la consommation énergétique, sur la base d'une opposition entre les approches expérimentales (en environnement réel) et celles effectuées dans un environnement simulé.

Le **chapitre 4** présente une étude de la consommation énergétique des communications en technologies cellulaires. Ce chapitre fournit des guides de bonnes pratiques en termes d'efficacité énergétique pour le développeur d'applications embarquées lors de l'utilisation de modules de communication cellulaire.

Le **chapitre 5** présente notre proposition d'un framework de profilage énergétique basé sur une approche matérielle pour la mesure de la consommation énergétique. Ce chapitre traite du fonctionnement de ce framework mais aussi de l'importance d'intégration de la consommation énergétique au cycle de développement traditionnel des applications embarquées.

Le **chapitre 6** présente un processus d'inférence de modèles énergétiques pour des périphériques embarqués à partir des traces de consommation. Ce chapitre décrit l'importance de tenir compte de la consommation énergétique des périphériques embarqués et la marche à suivre afin de fournir des modèles résumant le comportement énergétique de chaque périphérique profilé.

Le **chapitre 7** conclut ces travaux de thèse, discute de leurs limites et de leurs manques, puis ouvre sur de nouvelles perspectives de poursuite.

1.4 Contributions

Les travaux de thèse ont donné lieu à diverses contributions :

— Expérimentation de l'état de l'art et mise en évidence des lacunes présentes dans les simulateurs énergétiques lors de leur utilisation pour un scénario caractéristique de l'Internet des Objets [CGVB15] [CGVB16] ;

— Conception d'un framework de profilage énergétique permettant la cartographie du code source d'un logiciel embarqué sur un objet connecté [CGBV16] ;

— Conception d'une plate-forme de mesure énergétique embarquée reproductible à bas-coût ;

— Proposition d'un processus d'inférence de modèles énergétiques pour les composants périphériques d'un système embarqué [CBV+17].

Première partie

État de l’art : recherche, challenges et expérimentations

Deuxième partie

État de l’art

Chapitre 2

État de l'art

Ah ! ce n'est pas dans la science qu'est le bonheur,
mais dans l'acquisition de la science !
Savoir pour toujours, c'est l'éternelle béatitude ;
mais tout savoir, ce serait une damnation de démon.
Edgar Allan Poe, Puissance de la parole (1845)

2.1 Introduction

La conception d'applications pour objets connectés dans le secteur de l'Internet des objets nécessite une attention particulière à leur consommation énergétique. Afin d'appréhender au mieux cette contrainte énergétique, le développeur a besoin de connaitre le profil de consommation énergétique de son application lors du développement. Le but premier, est de détecter les points chauds de consommation énergétique du logiciel afin de les corriger et de les optimiser.

La mesure et l'estimation de la consommation énergétique d'un logiciel embarqué sur un objet communicant peuvent être effectuées via plusieurs méthodes et approches qui ont été proposées par les secteurs de la recherche et de l'industrie. Ces méthodes se différencient entre elles selon plusieurs critères tels que le coût, le caractère invasif ou bien encore la précision de l'estimation ainsi que d'autres critères que nous détaillerons plus amplement dans la suite de ce chapitre.

Il peut donc s'avérer extrêmement difficile de choisir dans la masse, une approche appropriée pour l'estimation de la consommation énergétique de son application embarquée sans la connaissance des différentes méthodes disponibles et leurs caractéristiques. D'autre part, cette connaissance de l'état de l'art est aussi nécessaire lors de l'amélioration d'une approche d'estimation énergétique existante ou la conception d'une nouvelle.

Nous présentons dans ce chapitre, une étude des différentes techniques existantes pour la mesure et l'estimation de la consommation énergétique d'un logiciel embarqué en les catégorisant selon différents critères.

2.2 La consommation énergétique d'un système embarqué : du transistor au logiciel

Notre sujet d'étude durant cette thèse est le profilage énergétique du code source d'applications embarquées pour systèmes contraints en énergie. Le choix de se centrer sur l'étude de la couche applicative et logicielle d'un système est dicté par l'envie de fournir au développeur d'applications embarquées une aide et une assistance directe sur la partie qui intéresse le plus ce dernier, le logiciel qu'il développe. Néanmoins, nous pensons qu'une compréhension précise de l'activité énergétique d'une application embarquée passe par l'appréciation du comportement énergétique du système dans toute sa verticalité.

Le logiciel applicatif d'un système embarqué gouverne et commande la partie matérielle de ce dernier. Néanmoins, c'est bien le circuit matériel qui est le responsable physique de la consommation énergétique du système. Ainsi, nous pensons qu'il est primordial d'intégrer les principales causes et sources physiques menant un circuit intégré matériel à consommer de l'énergie. Nous nous proposons donc de parcourir l'origine de cette consommation au fil de cette section. Enfin, nous présenterons les moyens physiques utilisés afin de mesurer cette consommation. text

2.2.1 Consommation énergétique d'un circuit intégré

Les circuits intégrés actuels sont pour l'immense majorité conçus avec des transistors en technologie CMOS (ang. *Complementary Metal Oxide Semiconductor*).

La transistor CMOS

Un transistor CMOS (c.f. Figure 2.1) est un composant semi-conducteur composé de trois broches nommées Source, Grille et Drain. Dans les circuites numériques, le transistor CMOS peut être vu comme un interrupteur commandé en tension. En fonction de la tension électrique appliquée sur la grille du transistor, ce dernier laissera ou non passer le courant entre la source et le drain. Plus précisément, chaque transistor CMOS est caractérisé par une valeur de tension appelée "*tension de seuil*" qui définira la tension minimale (ou maximal selon le canal N/P) requise afin de faire commuter le transistor.

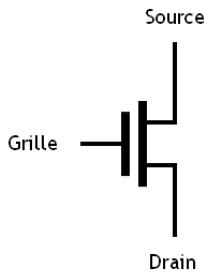


FIGURE 2.1 – Schéma d'un transistor CMOS

Nous trouvons deux types de transistors :

— les transistors NMOS qui deviennent passants (i.e. laisse passer le courant électrique) si la tension appliquée sur la grille est supérieure à la tension de seuil ;

— les transistors PMOS qui à l'inverse des NMOS deviennent bloquant (i.e. bloque le courant électrique) si la tension appliquée est supérieure à la tension de seuil.

La composition d'un transistor CMOS est d'une grande simplicité (figure 2.2). Nous trouvons ainsi à l'intérieur un bout de semi-conducteur, de type P ou N selon que cela soit un transistor NMOS ou PMOS, reliant la source et le drain. Sur ce semi-conducteur est placé une couche d'isolant le séparant ainsi de l'armature représentée par une simple électrode métallique reliée à la grille.

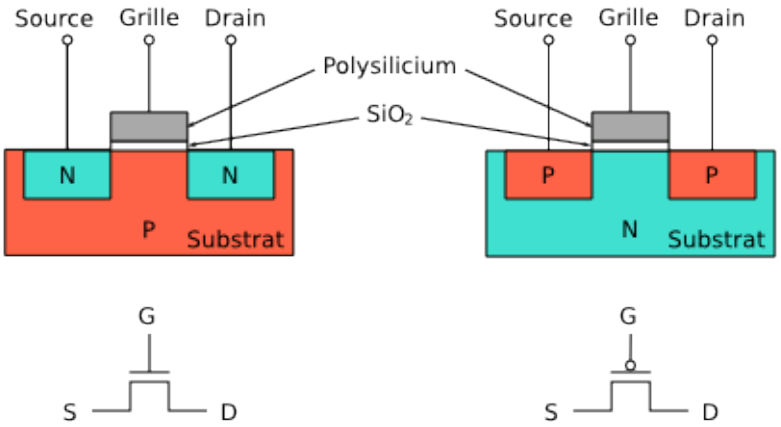


FIGURE 2.2 – Composition interne d'un transistor CMOS

De même que sa composition, le fonctionnement d'un transistor CMOS est assez simple (figure 2.3). Dans le cas d'un transistor PMOS, lorsqu'une tension est appliquée sur la grille, l'armature de la grille se charge en électron (i.e. négativement). L'accumulation des électrons sur la grille va engendrer un effet de répulsion avec les électrons circulant dans le semi-conducteur lorsqu'un courant est établi entre la source et le drain. À partir d'une certaine tension (*i.e.* Tension de seuil), l'effet de répulsion sera assez puissant afin de totalement bloquer la circulation du courant et ainsi rendre le transistor non passant.

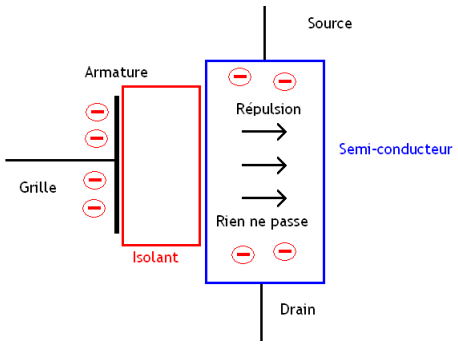


FIGURE 2.3 – Fonctionnement interne d'un transistor CMOS (de type PMOS ici)

Consommation énergétique de la technologie CMOS

La consommation énergétique des circuits intégrés à base de technologie CMOS est très souvent subdivisée en deux partie distinctes : une consommation énergétique dynamique et une consommation statique (equation 2.1).

$$P_{totale} = P_{dynamique} + P_{statique} \tag{2.1}$$

Afin d'illustrer au mieux ce phénomène de dissipation énergétique dans un circuit CMOS, nous prenons comme exemple un inverseur CMOS (Figure 2.4) qui représente la plus petite entité logique d'un circuit intégré CMOS. Il est composé d'une mise en série d'un transistor PMOS et d'un transistor NMOS. Dans cette illustration, nous modélisons le prochain étage du circuit via un condensateur de capacité C_{ch}. V_e et V_s représentent respectivement la tension d'entrée appliquée sur la grille et la tension de sortie, tandis que V_{dd} représente la tension nominale du circuit.

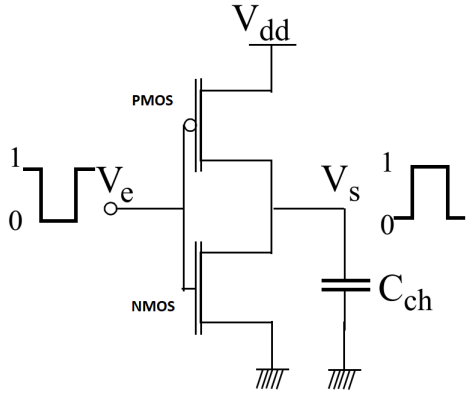


FIGURE 2.4 – Inverseur CMOS composé de deux transistors, un PMOS et un NMOS

La consommation énergétique dynamique :

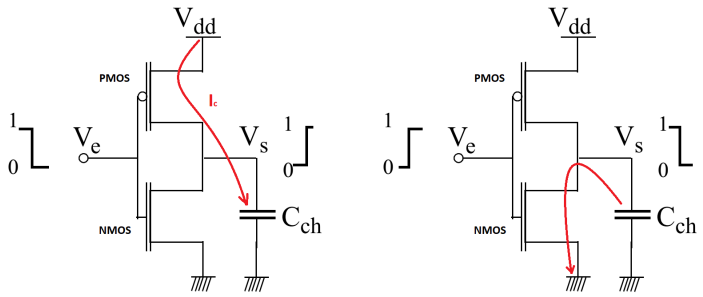
La consommation énergétique dynamique d'un circuit intégré correspond à la consommation due à l'activité du circuit. Autrement dit, l'énergie dissipée lors de l'accomplissement de différentes tâches. D'un point de vue matériel, cela se traduit par le changement d'état (*i.e.* commutation) des transistors constituant ce circuit.

Cette consommation dynamique est divisée en deux composantes : la consommation relative au courant de court-circuit et la consommation liée au courant de commutation.

i) Courant de court-circuit : Ce courant correspond à une dissipation énergétique préjudiciable pour le circuit. Il se produit lors d'une transition montante de la tension d'entrée, dans le cas d'un inverseur CMOS. Cela provoque la commutation au même moment des deux transistors PMOS et NMOS et durant un très court laps de temps, un court-circuit se crée entre la masse et l'alimentation du circuit dissipant ainsi de l'énergie. Ce courant de court-circuit est proportionnel au temps de transition et il reste souvent très faible et de plusieurs ordres de grandeur inférieur par rapport au courant de commutation.

ii) Courant de commutation : Le courant de commutation représente la cause principale de dissipation énergétique dans un circuit CMOS. Ce courant est directement dû à la commutation de l'inverseur. Si nous

reprenons la modélisation de l'inverseur CMOS illustrée à la figure 2.4, les commutations de ce dernier vont engendrer alternativement la charge et la décharge de la capacité C_{ch} en sortie. Lors de ces cycles, de l'énergie est dissipée (cf. Figure 2.5).



(a) Inverseur CMOS lors d'une transition descendante

(b) Inverseur CMOS lors d'une transition montante

FIGURE 2.5 – Inverseur CMOS lors d'une transition descendante et montante

Lors d'une transition descendante en entrée de l'inverseur (cf. Figure 2.5a), le transistor PMOS devient passant et le transistor NMOS bloquant. Ainsi, la capacité C_{ch} en sortie se charge via un courant de charge I_c. Durant cette transition, l'alimentation fournit une puissance électrique équivalente à $P_{alim}(t)$ (equation 2.2). L'énergie résultante sera donc égale à E_{alim} (i.e. équation 2.3) :

$$P_{alim}(t) = V_{dd}\, C_{ch} \frac{dV_s}{dt} \tag{2.2}$$

$$E_{alim} = \int_0^{T_{transition}} P_{alim}(t)dt = C_{ch}V_{dd}^2 \tag{2.3}$$

Néanmoins, l'énergie stockée $E_{C_{ch}}$ dans la capacité C_{ch} équivaut uniquement à la moitié de l'énergie fournie par l'alimentation (cf. 2.4), l'autre moitié étant dissipée dans les transistors par effet joule.

$$E_{C_{ch}} = \int_0^{T_{transition}} P_{C_{ch}}(t)dt = \frac{1}{2}C_{ch}V_{dd}^2 \tag{2.4}$$

Enfin, lors d'une transition descendante de l'inverseur (cf. Figure 2.5b), la capacité C_{ch} se décharge entièrement. De ce fait toute l'énergie ($\frac{1}{2}\, C_{ch}\, V_{dd}^2$) est dissipée. Nous pouvons ainsi déduire la puissance lié au courant de commutation $P_{commutation}$ pour un circuit intégré CMOS (equation 2.5. $\alpha \in [0-1]$ étant le nombre moyen de commutation du composant (i.e. inverseur ici) CMOS et $F_{horloge}$ représentant la fréquence d'horloge cadençant le circuit intégré CMOS.

$$P_{commutation} = \alpha\, F_{horloge}\, C_{ch}\, V_{dd}^2 \tag{2.5}$$

En supposant une architecture CMOS à n composants inverseurs et en négligeant le terme P_{cc} correspondant au courant de court-circuit, nous pouvons donc estimer la puissance énergétique dynamique de cette architecture $P_{dynamique}$ comme suit :

$$P_{dynamique} = \sum_{i=0}^{n} \alpha_i F_{horloge} C_{ch_i} V_{dd}^2 \tag{2.6}$$

De part cette courte étude, nous pouvons observer que la consommation énergétique dynamique d'un circuit intégré est influencée par la fréquence d'horloge de ce circuit, sa tension nominale d'alimentation ainsi que le nombre de cellule CMOS présentes dans le circuit et leur nombre de commutations. La fréquence du circuit joue ainsi un rôle important (de même pour V_{dd}) dans la dissipation énergétique d'un circuit CMOS.

La consommation énergétique statique :
La consommation énergétique dynamique présentée précédemment représente la majeur partie de la consommation d'un circuit intégré CMOS. Néanmoins, il existe aussi une consommation statique se produisant lorsque le circuit est inactif et n'effectue aucune tâche.

La consommation statique est provient des courants de fuite. Ces courants traversent de manière opportune la barrière (i.e. isolant) se trouvant entre la grille et le semi-conducteur d'un transistor. Cela induit une dissipation énergétique non volontaire et non productive. L'alimentation doit alors remplacer les charges perdues (i.e. ayant fuit) au niveau de la grille du transistor.

Avec l'évolution des technologies de gravure et la miniaturisation des transistors, cette couche d'isolant est devenue de plus en plus fine, laissant ainsi une plus grande liberté aux courants de fuite. Malgré cela, la consommation énergétique statique reste encore négligeable par rapport à la consommation dynamique d'un circuit intégré CMOS.

2.2.2 Comment mesurer la consommation énergétique ?

Nous avons pu voir dans la section précédente les principaux phénomènes physiques qui sont à l'origine de la dissipation énergétique et donc de la consommation énergétique, dans les circuit intégrés CMOS.

La consommation énergétique dans les circuit intégrés étant maintenant brièvement présentée, nous pouvons nous demander comment mesurer ou quantifier physiquement cette consommation ?.

««««< HEAD Dans notre situation, le cas du petit embarqué ou embarqué profond (ang. *deep embedded*, nous pouvons avancer que les choses sont différentes et même, sur certain points, un peu simplifiées. En effet, la grande majorité des plateformes matérielles embarquées fonctionne avec une tension (i.e. Voltage) d'alimentation fixe. Par ailleurs, Même si nous trouvons de plus en plus de plateforme matérielles équipées de processeurs embarqués proposant des schémas de fonctionnement avec une tension variable, le nombre de valeur que peut prendre celle-ci reste très restreint par rapport à ce qui est démocratisé sur les processeurs grand public. Ainsi, dans notre cadre de recherche, la mesure de la consommation énergétique d'une plateforme matérielle revient très souvent à l'étude et l'évaluation de l'intensité du courant électrique. Nous rappelons d'ailleurs à cet effet que l'énergie consommée E est simplement définie par l'intensité moyenne du courant, la tension (qu'on prend comme fixe) ainsi que le temps : $E = I_moy * V * T$. Le voltage étant fixe et le temps étant une métrique trivial à évaluer, le vrai verrou technique concerne la mesure de l'intensité du courant circulant dans la plateforme. Parmi les problématiques liées à cette mesure, nous pouvons citer la grande variabilité de cette grandeur dans les petit systèmes embarqués, ou encore les difficultés liée à la non perturbation du fonctionnement du système embarqué lors de l'accomplissement de la dite mesure. ======= Dans notre situation, le cas du petit système embarqué ou embarqué profond (ang. *deep embedded*), nous pouvons avancer que les choses sont différentes et même, sur certain points, un peu simplifiées. En effet, la grande majorité des plateformes matérielles embarquées fonctionne avec une tension d'alimentation fixe. Par ailleurs, même si nous trouvons de plus en plus de plates-formes

matérielles équipées de processeurs embarqués proposant des schémas de fonctionnement avec une tension variable, le nombre de valeurs que peut prendre celle-ci reste très restreint par rapport à ce qui est démocratisé sur les processeurs grand public, plus puissants. Ainsi, dans notre cadre de recherche, la mesure de la consommation énergétique d'une plate-forme matérielle revient très souvent à l'étude et l'évaluation de l'intensité du courant électrique. Nous rappelons d'ailleurs à cet effet que l'énergie consommée E est simplement définie par l'intensité moyenne du courant, la tension (qu'on prend comme fixe) ainsi que le temps : $E = I_moy * V * T$. Le voltage étant fixe et le temps étant une métrique trivial à évaluer, le vrai verrou technique concerne la mesure de l'intensité du courant circulant dans la plateforme. Parmi les problématiques liées à cette mesure, nous pouvons citer la grande variabilité de cette grandeur dans les petit systèmes embarqués, ou encore les difficultés liée à la non perturbation du fonctionnement du système embarqué lors de l'accomplissement de la dire mesure. »»»> 02a973c2d8fc36982ed04c66a03f6c5ee46f5257 ======= Dans notre cas d'étude spécifique, celui des petits systèmes embarqués (ang. *deep embedded*), nous pouvons avancer que la procédure de mesure reste traditionnelle. En effet, la grande majorité des plates-formes matérielles embarquées fonctionne avec une tension d'alimentation fixe. Par ailleurs, même si nous trouvons de plus en plus de plates-formes matérielles équipées de processeurs embarqués proposant des schémas de fonctionnement avec une tension variable, le nombre de valeurs de tension que peuvent prendre ces plates-formes restent très restreintes par rapport à ce qui est démocratisé sur les processeurs grand public, plus puissants. »»»> 468305393ff4e54ce221af07efd6845f6a2f2d09

Dans notre cadre de recherche, la mesure de la consommation énergétique d'une plate-forme matérielle revient très souvent à l'étude et l'évaluation de l'intensité du courant électrique. À cet effet, l'énergie consommée E est simplement définie par l'intensité moyenne du courant, la tension ainsi que le temps : $E = I_{moy} * V * T$.

La tension étant fixe, le vrai verrou technique concerne la mesure de l'intensité du courant circulant dans la plate-forme. Parmi les problématiques liées à cette mesure, nous pouvons citer :

- la grande variabilité de cette grandeur dans les petit systèmes embarqués ;
- les difficultés liée à la non perturbation du fonctionnement du système embarqué lors de l'accomplissement de la mesure.

Résistances de shunt

Afin de mesurer l'intensité d'un courant électrique, l'une des approches les plus couramment choisie est l'utilisation d'une résistance de shunt (ang. *shunt resistor*). Cette technique consiste à placer une résistance en série par rapport à la plate-forme matérielle dont on veut mesurer la consommation énergétique et l'unité d'alimentation de celle-ci. Au vu du rôle et du placement de la résistance, celle-ci est souvent choisie afin qu'elle soit d'une haute précision et d'une faible impédance, afin de limiter au plus la perturbation et l'interférence du système embarqué à mesurer.

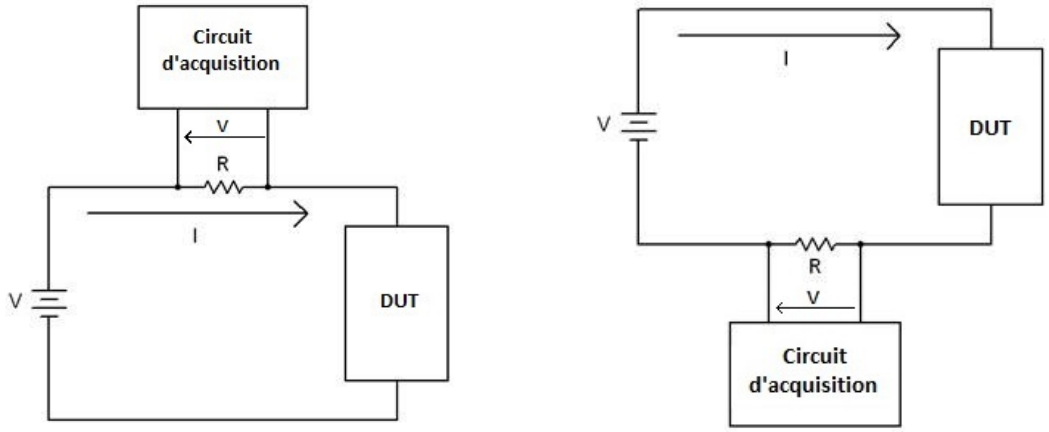
Une fois cette approche mise en place, l'intensité du courant circulant dans le système embarqué peut simplement être retrouvé en utilisant la loi d'Ohm ($U = R * I$). En effet, la mesure du courant équivaut ainsi à une mesure de la tension aux bornes de la résistance.

Le placement de la résistance de shunt en série par rapport au système à mesurer (i.e. DUT [1]) ne définit pas l'ordre de câblage. Il y a deux positions possibles :

1. DUT : Device Under Test

— résistance placée avant le système embarqué à mesurer (ang. *High-Side Current Measurements*) : Dans ce cas de figure, la résistance est sur le chemin d'alimentation du circuit de charge (figure 2.6a). L'avantage de cette disposition est la couverture complète des fluctuations de courant existantes. En d'autre terme, cela assure de détecter tous les courants venant du circuit de charge et donc les possibles fuites. Le désavantage de cette méthode est le risque de présence d'une tension en mode commun au borne de la résistance. Cette dernière pouvant être assez élevée, des risques d'endommagement du circuit de d'acquisition (i.e. de mesure) peuvent exister. Néanmoins, dans le cas des petits systèmes embarqués, les tensions rentrantes en jeu sont trop basses afin de présenter un tel risque ;

— résistance placée après le système embarqué à mesurer (ang. *Low-Side Current Measurements*) : La résistance de shunt est positionnée ici sur le chemin de retour du courant de charge (i.e. après passage du courant dans le système à mesurer) (figure 2.6b). Inversement au cas cité ci-dessus, ce dispositif protège le circuit d'acquisition contre toute tension en mode commun. Néanmoins, les fuites de courant existantes seront difficiles à capturer de par le circuit de mesure.



(a) *High-side* (b) *Low-side*

FIGURE 2.6 – Mesure via résistance de shunt

La disposition de la résistance fixée, la tension aux bornes de celle-ci permet de retrouver l'intensité du courant consommée. Afin de mesurer et collecter cette tension, un dispositif de type CAN (i.e. Convertisseur Analogique Numérique ou en ang. *ADC* pour *Analog to Digital Converter*) doit être utilisé. Cependant, du fait de la faible impédance des résistances de shunt utilisées, la tension mesurée est très souvent faible. Il est donc complexe voire impraticable de pouvoir facilement la quantifier. L'ajout d'un étage d'amplification aux bornes de la résistance de shunt via, le plus souvent, un amplificateur opérationnel est une solution des plus utilisée pour l'obtention d'une tension facilement quantifiable et donc mesurable.

Autres méthodes

Mise à part la résistance de shunt, plusieurs autres techniques existent pour la mesure matérielle du courant/énergie consommé dans un circuit intégré. Nous pouvons citer la *sonde de courant à effet hall* qui se base sur la mesure d'un champ magnétique. La sonde mesure le champ magnétique produit par la circulation d'un courant électrique dans un conducteur et lui associe une tension. Par Effet Hall, la valeur de cette tension est proportionnelle à l'intensité du courant qui circule dans le conducteur. Il est ainsi possible de calculer facilement cette intensité et donc la consommation énergétique du circuit. L'un des inconvénients majeurs de cette technique est la grande sensibilité du dispositif à des facteurs externes pouvant perturber le champ magnétique mesuré ainsi que le coût effectif de la sonde.

Une autre méthode différente pour la mesure de l'énergie consommée dans un circuit intégré consiste en l'utilisation de condensateurs. Elle se présente par l'utilisation dans le circuit de mesure, d'un ou de plusieurs condensateurs de capacité connue. Ces condensateurs se chargent et se déchargent à une fréquence proportionnelle à l'activité énergétique du circuit mesuré. Ainsi, le comptage du nombre de cycles de charge/décharge permet de donner une estimation de l'énergie consommée. Le problème de cette méthode réside dans la difficulté du choix et du calibrage des condensateurs selon la dynamique du courant mesuré.

Conclusion

En conclusion, parmi diverses techniques matérielles existantes pour la mesure de la consommation énergétique d'un circuit intégré, nous retenons l'utilisation des résistances de shunt comme la méthode offrant le meilleur compromis entre la facilité de mise en œuvre, le coût et la précision de la mesure.

2.3 Profilage d'une application embarquée

Un système embarqué est souvent vu comme la plate-forme matérielle effectuant physiquement les tâches pour lesquelles il a été conçu. Néanmoins, à l'image d'une voiture, un système matériel dans notre cas, n'est rien sans son pilote. Le logiciel embarqué endosse ici, le rôle de pilote pour le système.

2.3.1 Introduction

Le développement de logiciels embarqués pour des petits systèmes contraints en énergie est une tâche complexe. Le développeur doit fréquemment faire face à divers facteurs freinant le développement. Parmi les plus significatifs, nous pouvons citer :

— un espace mémoire réduit ;
— une unité de traitement de faible puissance ;
— une source énergétique limitée.

Ainsi, tout au long du cycle de développement du logiciel embarqué, des décisions et des choix cruciaux doivent être réalisés afin d'atteindre le meilleur compromis entre performances, objectifs métiers et ressources disponibles. Dans tout les cas, l'objectif du développeur est d'optimiser les capacités de l'objet, tout en garantissant une durée de vie et un coût raisonnable. Les travaux académiques autant qu'industriels ont déjà mené

au développement de nombreux outils pour analyser la charge mémoire et l'utilisation du processeur dans une application embarquée.

Cependant, le problème de la contrainte énergétique ne peut être pris en compte avec les mêmes outils de haut niveau. La simple mesure de la consommation énergétique d'un système est bien plus complexe que la mesure du temps processus ou encore de l'occupation mémoire. Autrement dit, le lien entre la métrique physique (la consommation d'énergie) et l'aspect logique (le logiciel embarqué) est difficile à retrouver et à quantifier.

À contrario, la mesure du temps d'exécution d'une tâche logicielle ou de sa charge mémoire est un fait simple à réaliser par le système lui-même en utilisant son propre référentiel (e.g. un compteur de cycle d'horloge pour la mesure du temps). Cette barrière physique, pousse les acteurs du monde des systèmes embarqués à rechercher des nouveaux outils pour le profilage énergétique des applications embarquées.

2.3.2 Intérêt pour le monde académique et industriel

La diversification des objets intelligents alimentés par batterie a entraîné une avancée majeure dans la recherche concernant l'optimisation de leur consommation énergétique. Nous avons vu, au cours de la dernière décennie, la publication d'un grand nombre de protocoles réseaux efficaces en énergie, de politiques d'économie énergétique et d'algorithmes de décision par rapport aux différentes sources énergétiques présentes [HC08] [Wes06] [PHC04] [Dun11]. Les différents composants matériels utilisés sur les plates-formes embarquées ont également évolués. Ainsi, il est devenu courant d'utiliser des microcontrôleurs ayant des architectures complexes dans la conception des systèmes embarqués. Par conséquent, la phase de profilage de la consommation énergétique de ces systèmes devient une nécessité tant pour les acteurs industriels que pour la recherche.

Dans le domaine de la recherche académique, l'illustration de la contrainte énergétique est le plus souvent limitée aux scénarios de réseaux ad-hoc sans-fil, autrement appelés réseaux de capteurs sans fil. Les besoins dans ce cas d'utilisation particulier sont principalement axés sur l'estimation de la durée de vie de l'ensemble du réseau, et le profilage énergétique distribué et synchronisé sur tous les nœuds du réseau. Ainsi, l'accent est mis sur l'observation des conséquences énergétiques d'une action d'un ou plusieurs nœuds sur l'ensemble du réseau. Cependant, les cas d'utilisation impliquant des réseaux de capteurs ne sont pas encore très répandus dans le monde industriel. En effet, le nombre de nœuds dans un réseau reste très limité, et se retrouve généralement égal à un unique objet connecté communicant directement avec une passerelle réseau.

Par conséquent, les exigences de profilage énergétique entre les des deux mondes que sont la recherche et l'industrie, se retrouvent totalement différentes. En général, les projets industriels sont liés à des clients par un accord de service (ang. *Service Level Agreement - SLA*), qui contraint l'industriel à assurer un certain niveau de qualité et d'adéquation au cahier des charges du client. Ainsi, dans le cas d'objets alimentés par batterie, une estimation précise de la durée de vie est requise dans la majorité des cas envisageables du scénario d'utilisation du client.

Le développeur a donc besoin d'un profil énergétique à grain fin de chaque activité de l'objet connecté, où chaque bloc du code source de l'application embarquée est relié à la contribution énergétique correspondante. Cela permettra de prédire avec une faible erreur, la durée de vie appropriée selon le type de source énergétique utilisée. Néanmoins, un des facteurs décisifs reste le surcoût causé par la phase de profilage énergétique. L'achat de matériels nécessaires au fonctionnement de l'approche de profilage représente pour l'industriel, une

faible part de ce surcoût. Le coût principal est dépendant du temps consacré à la solution de profilage afin de l'adapter aux besoins du développement envisagé, et de l'utiliser. Dans cette optique, les fonctionnalités d'aide à un développement efficient en énergie, ou la flexibilité de conception de la solution, sont des conditions fondamentales pour les acteurs industriels d'aujourd'hui.

2.4 Les types d'approches de profilage énergétique d'un logiciel embarqué

Nous présentons dans cette section les critères d'évaluation d'un profileur énergétique ainsi que notre classification des approches de profilage énergétique existantes.

2.4.1 Critères de sélection et d'appréciation des approches

Dans cette partie, nous discutons des principaux critères et références requis pour un profileur énergétique. Nous sélectionnons et présentons les critères les plus efficaces que chaque approche de profilage énergétique devrait viser à atteindre afin d'arriver à un niveau acceptable de qualité d'estimation énergétique.

Estimation de la durée de vie

Dans la plupart des cas d'utilisation, les objets connectés sont conçus afin d'effectuer des opérations sur une longue période, allant de quelques semaines à plusieurs années. Les réseaux de capteurs traditionnels représentent le meilleur exemple de cela. Nous pouvons généraliser cet aspect en admettant que tout objet connecté tend généralement à réaliser diverses fonctions pour une période de temps spécifique (semaines, mois ou années). En outre, il existe de nombreux cas d'utilisation où le rechargement de la batterie se trouve être difficile voire même totalement impossible (un objet enfoui dans le béton pour la surveillance de structures).

En conséquence, l'estimation ou la prédiction de la durée de vie énergétique, peut être considérée comme l'exigence principale de chaque approche de profilage énergétique. En d'autre termes, nous pouvons énoncer la problématique de l'estimation de la durée de vie comme suit :
Quelle est la durée de vie moyenne qu'un objet peut atteindre, avec un stock d'énergie initial défini et la réalisation de certains types d'activités, en tenant compte de l'environnement de déploiement de l'objet ?

Gestion en ligne de la consommation énergétique

Après leur déploiement effectif, les objets connectés peuvent parfois fonctionner en total autonomie, hors de contrôle de tout utilisateur ou administrateur pendant de courtes ou longues périodes. En raison de l'environnement de déploiement qui peut exposer l'objet, à des conditions incertaines, le nombre total d'activités accompli par l'objet ne peut être déterminé avec précision. En outre, parmi les différentes activités, certaines peuvent avoir une priorité bien plus élevée que d'autres. Ces dernières doivent donc avoir l'assurance de pouvoir être exécutées à tout moment (e.g. une activité de communication d'urgence). L'objet doit donc être capable d'ajuster dynamiquement l'exécution des ses activités, en fonction de l'environnement, mais surtout de l'énergie restante en réserve (cf. batterie ou récupération d'énergie).

L'outil de profilage peut avoir deux fonctions dans ce critère. La première est la génération de modèles énergétiques pouvant être utilisés sur l'objet lui même après déploiement afin d'ordonnancer l'exécutions de ses tâches. En seconde fonction, l'outil de profilage peut être déporté en partie sur l'objet, afin de fournir à celui-ci la capacité de mesurer après déploiement la consommation énergétique de ses tâches. L'objet pourra ainsi tracer la consommation énergétique en fonction des différentes périodes de sa vie en déploiement.

La problématique liée à ce critère peut donc être résumée de cette façon :
Un profileur de consommation énergétique pour système embarqué prenant en compte ce critère est capable de fournir à l'objet des fonctions, lui permettant de gérer dynamiquement et de manière autonome sa consommation énergétique, après son déploiement effectif.

Granularité la mesure de la consommation énergétique

L'estimation de la consommation énergétique peut avoir de multiples significations. Vu de manière globale, l'estimation de la durée de vie énergétique d'un objet est appropriée lorsque l'on considère le profil énergétique d'un ensemble d'activités de l'objet. Cependant, un développeur embarqué peut souvent souhaiter obtenir des chiffres énergétiques plus précis sur son propre code source embarqué. Cela lui permet d'identifier avec précision, la consommation énergétique de plusieurs portions particulières de son code source.

Le paramètre décrivant cette précision est la granularité ou résolution de l'estimation et du profilage énergétique. La granularité offerte par un outil de profilage énergétique peut varier selon le niveau de profondeur atteignable lors de l'analyse énergétique d'un code source embarqué. Un développeur pourrait donc obtenir des informations de consommation d'une application à plusieurs niveau de granularité (fonctionnel, bloc fonctionnel, bloc de base, instruction, etc.).

Ce critère d'évaluation peut donc être résumé de la façons suivante :
À quel niveau de grain, un outil de profilage énergétique est capable de fournir des chiffres de consommation énergétique ?.

Assistance au développement efficace en énergie

Les précédents critères définissent l'efficacité technique d'un outil de profilage énergétique. Néanmoins, le but premier et réel de tels outils est d'aider le programmeur lors du développement d'applications embarquées soumises à des contraintes énergétiques. En particulier, la façon dont les résultats de consommation énergétique d'une application sont présentés au développeur, ont un impact considérable sur l'efficacité de ce dernier à rapidement corriger les bugs énergétiques présents.

Un affichage brut des chiffres de consommation, sans pré-traitement préalable, peut submerger le développeur sous un amas d'informations inutiles, lui rendant sa tache plus ardus. En outre, certains profileurs énergétiques peuvent aller au delà de la phase de présentation des résultats, en apportant une assistance de correction du code source de l'application. Ainsi, des profileurs peuvent suggérer au programmeur des astuces pour un développement efficace en énergie, à partir des résultats de profilage et d'une base de données de motifs de code source identifiables.

Ce critère peut être résumé comme suit :
Quels sont les fonctionnalités mises en évidence par l'outil de profilage énergétique, afin d'améliorer et de faciliter

la compréhension du développeur, en regard des chiffres bruts d'estimation énergétique de son application embarquée.

Coût

Enfin, une caractéristique qu'on ne peut négliger concerne le coût global de l'outil de profilage. Ce coût peut être divisé en deux parties principales. La première, à laquelle on pense le plus naturellement, représente le coût d'installation de l'outil. Nous entendons par cela, le prix effectif des composants matériels nécessaires au fonctionnement de l'outil de profilage. Cependant, le coût global ne s'arrête pas là, et un second sous-critère correspondant à l'adaptabilité et la maintenance de l'outil est à prendre en compte. Ainsi, le temps passé à mettre en place et à adapter la solution aux exigences et besoins du développeur, représente un paramètre important influant directement sur le coût global de la solution de profilage. L'effort requis pourrait donc être inacceptable pour un projet industriel où le budget reste un élément critique.

La question à laquelle on doit répondre concernant ce critère est donc :
Quelle est la charge de travail devant être consacrée„ pour adapter l'approche de profilage énergétique afin de satisfaire les exigences souhaitées sur un projet donné ?

2.4.2 Classification des approches de profilage

La consommation énergétique des petits systèmes embarqués communicants est un phénomène difficile à mesurer [JDCS07b]. On peut globalement diviser les approches de mesure et d'estimation de la consommation énergétique en trois différentes catégories : les méthodes de mesure matérielles, les méthodes de mesure logicielles et les méthodes basées sur la simulation de systèmes embarqués.

Approches à base matérielle

Les approches de profilage matérielles peuvent être définies comme toutes les méthodes utilisant un moyen de mesure matériel afin d'estimer la puissance énergétique effective dissipée par le système cible en cours d'exécution. Fondamentalement, la consommation en énergie de chaque système (embarqué ou non) est décrite par la relation mathématique suivante : $E = \int_{t1}^{t2} u(t).i(t)dt$. Ainsi, l'estimation de la consommation énergétique du système embarqué se résume à mesurer $i(t)$ durant l'expérimentation ($u(t)$ est souvent fixe).

Cette mesure peut être effectuée de différente façons. La mesure via une résistance de shunt est certainement la plus connue et la plus utilisée. Celle-ci a été présentée au début de ce chapitre. L'acquisition des mesures de tension est réalisée à l'aide de dispositifs basés sur des convertisseurs analogiques numériques (ADC).

Enfin, les exigences sur les caractéristiques de la mesure varient selon l'usage et les objectifs du profilage. Ainsi, une résolution temporelle élevée de la mesure n'est pas forcement nécessaire, si la tâche de profilage énergétique ne se concentre que sur des fonctions de haut niveau du code source de l'application embarquée. Sur un autre aspect, le comportement énergétique d'un système embarqué peut afficher une large disparité dans les niveaux de consommation. De ce fait, le dispositif de mesure du profileur énergétique doit gérer une large dynamique de consommation de courant, allant de quelques microampères à plusieurs milliampères.

Approches à base logicielle

Au contraire des approches de profilage basées sur le matériel, les techniques basées sur le logiciel ne nécessitent, en principe, aucun matériel supplémentaire pour estimer la consommation énergétique d'un système embarqué. Cependant, à l'instar de la catégorie précédente, les méthodes basées sur le logiciel sont caractérisées par une estimation effectuée lors de l'exécution effective de l'application sur le système cible. Ce type de méthode tend à instrumenter ou à modifier le logiciel embarqué afin d'extraire différents types de caractéristiques ou de compteurs lors de l'exécution. Afin d'illustrer, nous pouvons citer la durée d'exécution d'un bloc de base du code source, ou encore le suivi du comportement de certains périphériques embarqués, en surveillant le code source responsable de l'activation ou désactivation de ce dernier. Les solutions basées sur ce type d'approche peuvent également profiter de toute sorte de compteur matériel existant sur le système embarqué cible. Ainsi, certains processeurs embarqués ou même périphériques externes exposent des statistiques sur leur utilisation et leurs états internes, pendant l'exécution. À titre d'exemple, la fréquence du CPU, l'état du périphérique (e.g. en veille, actif), la puissance d'émission d'un module radio, sont des statistiques souvent utilisées.

En tirant avantage de tous les chiffres rassemblés lors de l'exécution de l'application embarquée, le profileur énergétique peut déduire, en ligne ou hors ligne, une estimation de la consommation énergétique de plusieurs parties du code source de l'application.

Approches basées sur la simulation

Cette dernière catégorie fait le choix d'estimer la consommation énergétique d'un système embarqué en simulant son comportement. Ce type d'approche, autrement nommé *simulateur énergétique*, s'appuie sur une modélisation de la plate-forme cible à profiler. Les modèles énergétiques correspondants sont conçus dans une phase antérieure à celle de la simulation. Le but du modèle est d'attribuer une valeur énergétique à chaque bloc fonctionnel de l'application embarquée cible.

Lorsque l'application passe par la phase de simulation, le simulateur énergétique reconnaît les motifs ou patterns des blocs fonctionnels et comptabilise la valeur de consommation énergétique correspondante prédéfinie dans le modèle, puis l'ajoute à la consommation globale de l'application. La principale différence entre les différents simulateurs énergétiques existants est la résolution de la modélisation énergétique considérée. Ainsi, certains simulateurs se basent sur une modélisation complète mais assez lourde de la consommation énergétique de chaque instruction assembleur de l'application embarquée. De plus, plusieurs autres paramètres peuvent être prise en compte tels que la consommation inter-instructions ou encore la surconsommation des transferts mémoires.

En revanche, d'autres simulateurs énergétiques abstraient les modèles énergétiques utilisés en groupant la consommation énergétique allouée à plusieurs unités du systèmes. Les périphériques externes intégrés à la plate-forme embarquée sont aussi souvent utilisés. Fréquemment, cela est réalisé en modélisant le comportement énergétique de ces périphériques sous forme de machine à états finis.

Enfin, les simulateurs énergétiques différent également selon les exigences initiales du projet de développement de l'application cible. En effet, un simulateur de haute précision implique l'utilisation de lourds modèles énergétique qui considèrent plusieurs états énergétiques de plusieurs blocs fonctionnels du code source de l'application. Ces simulateurs peuvent d'ailleurs aussi avoir recours à des modèles prenant en compte et

Type	Approches de profilage
Profileurs matériels	ICount et Quanto [JDCS07b] [FDLS08], FlockLab [LFZ+13] PEEK [HJE+14], Nemo [ZX13], SPOT [JDCS07a] Energy Bucket [AH09], GoldCaps [RSV+05] Energy Endoscope [SMK08], IAR I-jet debugger [LF14] Atmel debugger [Cor16], MAGEEC board [Hol13]
Profileurs logiciels	PowerTrace [DEFT11] eProf Mobile Profiler (eProfMP) [PHZ12] [PHZ+11] Eprof [SKZS12], PowerScope [FS99], AppScope [YKJ+12] Eon [SKG+07], ECOSystem [ZELV02] [ZELV03] [EL05]
Simulateurs	PowerTOSSIM [SHCW04] eSimu [FFF09] [FFF07] AEON [LWG05] Cooja [EÖV+09] [ÖED10]

TABLE 2.1 – Catégorisation des approches de profilage énergétique étudiées dans notre état de l'art.

intégrant de nombreux facteurs externes, qui affectent la consommation énergétique du système. Ces facteurs peuvent être environnementaux à l'image des bruits externes (e.g. radio), ou des facteurs liés aux perturbations au niveau de la plate-forme matérielle du système (e.g. dérive des horloges embarquées). Ainsi, la complexité du simulateur énergétique et ses caractéristiques en termes de performance sont proportionnels à la lourdeur du modèle énergétique utilisé.

2.5 État de l'art des approches de profilage énergétique

Après avoir défini et décrit les catégories d'approches de profilage existantes ainsi que leurs critères d'évaluation, nous décrivons dans cette section diverses méthodes de profilage énergétique, que nous considérons comme les plus significatives et fondamentales dans l'état de l'art associé à cette thématique. Pour chaque classe d'approche, nous détaillons les méthodes ayant des caractéristiques particulières et nous en résumons les autres dans un format plus raccourci.

Le tableau 2.1 décrit la liste de toutes les approches de profilage prise en compte dans notre état de l'art.

2.5.1 Les approches matérielles

ICount et Quanto

Dans [JDCS07b], les auteurs décrivent ICount, un circuit matériel simple et peu coûteux qui permet d'augmenter la plate-forme matérielle cible avec des capacités de mesure énergétique. ICount requiert de l'objet cible la présence d'une unité d'alimentation incluant une régulateur de tension DC-DC, en plus d'un compteur matériel dédié. Ce compteur est relié au régulateur et compte les cycles de commutation de celui-ci. Les auteurs démontrent que le nombre de cycles de commutation du régulateur est linéairement dépendant de l'énergie fournie par l'unité d'alimentation, et donc l'énergie consommée par le système cible. De ce fait, le circuit ICount donne la possibilité au système de mesurer sa propre consommation énergétique. Cette particularité peut permettre au système de prendre, par sa propre initiative, des décisions énergétiques après déploiement.

Dans [FDLS08], les auteurs décrivent Quanto, un composant logiciel directement intégré à l'application embarquée du système cible. Quanto exploite le compteur énergétique embarqué ICount pour effectuer la mesure de courant. Il instrumente les pilotes des périphériques intégrés au système afin de suivre la consommation énergétique des composants matériels et logiciels. Quanto définit une nouvelle abstraction nommé *activité*. Cette dernière permet à un développeur de caractériser, selon ses propres exigences, dans le code source applicatif, des fonctionnalités haut niveau de l'objet (e.g. une activité de transmission de données ou d'échantillonnage d'un capteur). Par ailleurs, les composants matériels (e.g. le CPU ou la radio) sont suivis pour tenir compte de leurs états fonctionnels et des activités auxquelles ils contribuent. Le module radio, par exemple, peut être dans différents états (transmission ou réception avec différents niveaux de puissance) et peut participer à plusieurs activités définies par l'utilisateur.

Les données collectées sont conservées dans la mémoire du système cible. Quanto effectue ensuite un traitement hors ligne (i.e. après exécution) de ces données. Il utilise pour cela une régression linéaire pondérée multivariée pour reconstruire la contribution énergétique de chaque composant matériel dans chaque activité, et celle de chaque activité dans la consommation globale. Le suivi des états énergétiques permet de reconstruire la causalité entre les activités. Le développeur est ainsi capable d'identifier les raisons qui ont conduit à une consommation énergétique donnée. En outre, le profileur a permis de confirmer que des plates-formes matérielles similaires pouvaient produire des chiffres de consommation énergétique différents [LMGF13]. Malgré les avantages évidents du module de mesure ICount, sa faible précision de $\pm 15\%$ le rend inadéquat pour un profilage de consommation à grain fin. Concernant Quanto, l'enregistrement des données de consommation énergétique directement dans la mémoire interne du système induit des phases de profilage de faible durée, en raison de la surcharge mémoire élevée.

FlockLab

Présenté dans [LFZ+13], FlockLab est un banc d'essai pour les systèmes embarqués communicants. Il n'est pas complètement centré sur le profilage énergétique, mais il l'expose comme étant une de ses fonctionnalités de base. La solution propose une plate-forme matérielle/logicielle qui vise principalement à faciliter le débogage simultané et synchronisé sur plusieurs nœuds communicants d'un réseau de capteurs.

La partie matérielle consiste en une plate-forme embarquant un microprocesseur nommée *observateur*. Chaque observateur peut profiler et déboguer jusqu'à quatre systèmes cibles simultanément. Néanmoins, en raison de l'hétérogénéité des systèmes embarqués communicants et du manque criant de standardisation, une carte d'interface matérielle est requise.

L'observateur échantillonne via l'ADC la tension au borne d'une résistance de shunt. En résolution complète (24 bits), l'ADC peut être échantillonné à une fréquence maximale de 24 kHz. Une caractéristique transversale de l'observateur permet de régler dynamiquement la tension d'alimentation de 1,8 V à 3,3 V, ce qui permet de simuler le comportement du système cible lors de la décharge de la batterie l'alimentant. FlockLab ne fournit aucun mécanisme afin d'attribuer la consommation énergétique au code source de l'application. Au lieu de cela, FlockLab s'appuie sur la disponibilité des broches GPIO [2] sur la plupart des systèmes, pour fournir un suivi énergétique et temporelle synchrone et distribué sur tous les systèmes cibles hébergés par un observateur. Les broches sont utilisées afin d'enclencher des événements logiques, et de marquer la courbe de consommation énergétique des systèmes cibles.

PEEK

PEEK (ang. *proactive energy-aware programming*) [HJE+14] est un framework qui vise à intégrer une méthode de développement efficace en énergie dans le cycle de conception de l'application embarquée. L'architecture de PEEK repose sur trois composants principaux : un front-end, un middle-end et un back-end.

Le front-end représente une interface entre le développeur et le back-end effectuant la tâche de mesure énergétique sur un code applicatif spécifique. Il permet d'envoyer des commandes, via le middle-end, au composant de mesure énergétique, et d'être notifié lorsque les résultats sont prêts. Le middle-end supervise une infrastructure basée sur des instantanés du code source de l'application. Pour réaliser cette mission, il s'articule autour du système de contrôle de révision Git. Le middle-end sert de file d'attente pour les tâches, commandes et les notifications, entre le front-end et le back-end.

Le back-end représente la plate-forme de mesure effective. Celle-ci se repose sur la conjonction de plusieurs condensateurs. Le courant à mesurer, charge et décharge alternativement deux condensateurs qui sont reliés à une bascule RS. La fréquence de commutation de la bascule a été démontrée comme directement proportionnelle au courant d'entré. Cette dernière est donc utilisée afin de calculer la consommation énergétique du système cible. Le principal inconvénient de cette méthode de mesure réside dans le bon choix d'un compromis entre la valeur des condensateurs et le niveau de courant fourni par l'alimentation.

Le framework PEEK s'oriente aussi sur l'aide au développeur embarqué. Il est ainsi capable de générer automatiquement plusieurs versions du code source de l'application embarquée et de tester différentes techniques de gestion de l'énergie (e.g. différents états de veille, DVFS(Dynamic Voltage and Frequency Scaling), etc.), afin de suggérer au développeur quel est le mécanisme le plus efficace en énergie à utiliser. Malheureusement, cette initiative innovante d'aide au développeur reste plutôt minimale, et n'est donc pas vraiment utilisable sur de vrais projets industriels. Elle représente tout de même une étape prometteuse pour la correction énergétique automatique du code source.

2. GPIO : General Purpose Input/Output

GoldCaps

Présentée dans [RSV+05], cette méthode de mesure repose sur une alternative à la mesure basée sur des résistances de shunt. L'estimation énergétique consiste ici à utiliser des condensateurs spécifiques appelés GoldCaps. Ces condensateurs sont caractérisés par une capacité bien plus grande (1 Farad) que les condensateurs traditionnels, souvent utilisés dans la conception de circuits intégrés. L'approche décrite se base sur l'utilisation de GoldCaps en remplacement de la batterie d'alimentation du système cible. Cela permet d'effectuer des expériences de courte durée (de plusieurs minutes à quelques heures), puis de déduire une durée de vie énergétique globale estimée pour le système cible, en fonction du logiciel embarqué.

Globalement, cette méthode souffre de plusieurs inconvénients, dont le temps limité des expériences de mesure pouvant être réalisées. En outre, la phase de mesure énergétique effective ne peut pas être véritablement corrélée avec le logiciel embarqué en exécution, ni par des événements logiques, ni par des événements temporels du système embarqué cible.

Autres travaux académiques

Outre les travaux précédents décrits avec détails, de nombreux autres ont choisi des techniques connexes afin d'atteindre l'objectif d'un profilage énergétique du code source.

Nemo

Un système de profilage énergétique non invasif nommé *Nemo* [ZX13] cible le domaine des réseaux de capteurs sans fil. Il se compose d'un circuit de mesure basé sur l'utilisation de résistances de shunt. Ce circuit est capable de s'adapter automatiquement en fonction de l'intensité du courant mesuré. Nemo a été conçu pour être intégré directement sur le système cible, offrant ainsi à ce dernier, la capacité de surveiller et mesurer *in situ* sa propre consommation. La communication entre le circuit de mesure et le système hôte s'effectue via un protocole half-duplex. Cependant, le circuit Nemo consomme environ 4,6 mA en mode actif et prend environ 2,3 s pour transmettre les données mesurées sauvegardées en mémoire, ce qui affecte alors le fonctionnement normal du système cible.

SPOT

SPOT décrit dans [JDCS07a] est un outil de profilage énergétique scalable ressemblant fortement à Nemo, utilisant d'ailleurs la même technique de mesure. SPOT est caractérisé par un taux d'échantillonnage de 1 MHz, ainsi qu'une fonction de résolution et adaptation dynamique au courant à mesurer. Les mesures effectuées sont stockées dans un compteur numérique et sont horodatées par un autre compteur. Un protocole de communication I2C est intégré pour permettre la lecture de ces compteurs par un système externe ou même le système cible. L'évaluation de SPOT indique une bonne précision de l'outil (i.e. 5% d'erreur en moyenne), néanmoins, il présente les mêmes inconvénients que Nemo.

Energy Bucket

Les auteurs décrivent dans [AH09] une plate-forme de mesure énergétique se reposant sur l'utilisation de condensateurs pour mesurer le courant drainé par un système embarqué. La plate-forme fournit à la cible une tension constante, et compte le nombre de cycles de charge et de décharge de plusieurs condensateurs combinés. Chaque cycle peut être comptabilisé avec une quantité fixe d'énergie, appelée "*bucketful*". Afin de transmettre les mesures capturées, une communication en série est nécessaire. Cela représente un vrai goulot d'étranglement lorsqu'une haute fréquence de lecture des mesures est nécessaire. Cela limite donc l'utilisation d'Energy Bucket à des mesures de longue durée où un profilage précis du code source n'est pas souhaité.

Energy Endoscope

Une plate-forme matérielle de profilage basée sur un FPGA, nommée *Energy Endoscope*, est présentée dans [SMK08] et [MHY+06]. La plate-forme se compose d'un circuit intégré dédié nommé LEAP2 (*ang.* Low Power Energy Aware Processing) qui est étroitement intégré au processeur hôte et à ses périphériques externes. Cette conception permet de mesurer avec précision la consommation de chaque sous-système du système cible avec une un échantillonnage approchant la milliseconde. Néanmoins, la solution proposée ne permet pas d'aller en profondeur dans le profilage énergétique afin de pouvoir proposer une corrélation entre consommation énergétique et code source exécuté. Enfin, l'intégration étroite entre les composants complexifie de manière significative l'utilisation de la solution sur un autre système cible basé sur une plate-forme matérielle différente.

Autres travaux industriels

Outre les travaux issus de la recherche académique, le monde industriel accorde aussi une grande attention afin de outrepasser le problème du profilage énergétique des systèmes embarqués. Plusieurs productions industrielles ont été proposées à cet effet. La sonde de débogage *IAR I-JET* [LF14] et la plate-forme de débogage énergétique Atmel [Cor16] sont deux solutions.

IAR I-jet debugger

La sonde IAR tire profit des puces ETM (*Embedded Trace Macrocell*) [Lim11] intégrée dans certains microcontrôleurs ARM, ou dans un mode dégradé, de la puce plus généralisée DWT (*Data Watchpoint and Trace Unit* [ISy15]). Lors du profilage énergétique, la sonde IAR alimente le système cible via une interface JTAG. Elle capture la consommation énergétique de cette dernière à l'aide d'un circuit de mesure basé sur une résistance de shunt. La puce de traçage et de suivi ETM (ou DWT) est requise afin d'obtenir un échantillonnage haute fréquence du compteur ordinal (*PC/Program Counter*) du microcontrôleur. En superposant de manière synchrone les chiffres énergétiques collectées et les échantillons du PC, l'IDE propriétaire IAR est en mesure de lier un bloc du code source avec une estimation de sa consommation énergétique. Le principale inconvénient de la sonde de profilage IAR réside dans l'absence des puces ETM et DWT dans les microcontrôleurs traditionnels. Enfin, le coût d'une telle sonde JTAG acceptant de haute fréquence d'échantillonnage est très significatif.

Atmel debugger

La proposition d'Atmel consiste en un co-design matériel/logiciel uniquement compatible avec les microcontrôleurs Atmel. À l'instar d'IAR, cette solution offre des fonctionnalités de débogage et de traçage via une interface JTAG. Elle est également basée sur une méthode de mesure via des résistances de shunt. Le profil énergétique et la corrélation avec le code source embarqué applicatif est effectué à l'aide d'une méthode d'échantillonnage statistique. Le registre du compteur PC du système cible est interrogé aussi rapidement que possible et la ligne de code source correspondante est directement associée à la courbe de consommation énergétique. Ainsi, des comportements inhabituels peuvent être détectés uniquement si un bloc du code source est suffisamment de fois appelé. La solution d'Atmel ne comporte donc pas de corrélation énergétique du code source à proprement dit. De plus, l'utilisation de cette solution n'est possible que sur des systèmes basés sur des microcontrôleurs Atmel.

MAGEEC board

Enfin, une plate-forme de mesure énergétique matérielle à faible coût a été conçue dans le cadre du projet industriel MAGEEC [Hol13]. La proposition est basée sur une plate-forme STM32F4 Discovery qui est augmentée avec un circuit de mesure matériel externe. Plusieurs résistances de shunt sont intégrées sur le circuit et sont sélectionnables manuellement afin d'adapter la mesure au niveau du courant en entré. En raison de la faible bande passante de l'interface de données USB utilisée, les seuls chiffres énergétiques en sortie sont les valeurs globales de consommation d'un intervalle de mesure spécifique. Cela représente donc un inconvénient important et pose une réelle restriction à la corrélation du code source embarqué et de sa consommation énergétique.

2.5.2 Les approches logicielles

PowerTrace

PowerTrace [DEFT11] est un profileur énergétique logiciel qui suit les états énergétiques du système cible, en mesurant le temps pendant lequel les composants matérielles (e.g. CPU, radio, flash, etc.) sont dans différents états. La consommation énergétique globale du système est simplement estimée comme la somme de toutes les consommations de tous les composants surveillés. PowerTrace se positionne comme une suite aux travaux déjà effectués sur l'estimation *in-situ* de la consommation énergétique dans [DÖTH07] et implémenté sur le système d'exploitation Contiki-OS [DGV04]. PowerTrace instrumente les pilotes logiciels des périphériques embarqués sur le système cible. Il insère ainsi une portion de code qui effectue un horodatage et est déclenché lorsqu'un périphérique matériel change d'état de fonctionnement. La consommation énergétique de chaque composant est calculée en utilisant la durée de chaque état et la puissance énergétique indiquée dans un modèle basé sur une simple *datasheet*.

Powertrace propose également le concept de *capsule énergétique*. Une capsule énergétique représente un ensemble d'états énergétiques et est associée à une ou plusieurs activités de haut niveau définies par le développeur (e.g. une transmission radio des donnés collectées via des capteurs). Par ailleurs, plusieurs capsules énergétiques peuvent être regroupées afin d'agréger la consommation énergétique de plusieurs activités de plus bas niveau. Enfin, chaque processus de Contiki-OS peut s'abonner à une capsule énergétique afin d'être informé en temps

réel de la consommation énergétique d'activités spécifiques. En plus d'une certaine simplicité de mise en œuvre (faible coût d'instrumentation logicielle), Powertrace offre de nombreuses fonctionnalités telles que le suivi de la consommation énergétique globale d'activités en temps réel. Cependant, Powertrace manque de précision lorsqu'il est question de profiler précisément et finement une activité. En outre, le processus d'estimation est entièrement basé sur la validité de la *datasheet* du système cible. Ce qui augmente la marge d'erreur [CGVB15].

eProf Mobile Profiler (eProfMP)

Présenté dans [PHZ12] et [PHZ+11], eProfMP est un profileur énergétique centré sur les applications mobiles pour smartphone. Il peut sembler inapproprié d'inclure cette solution dans notre étude axée sur les petits systèmes embarqués. Cependant, divers mécanismes de cette approche pourraient être appliqués pour évaluer la consommation énergétique des petits objets. La solution considère une hypothèse du monde réel, selon laquelle la plupart des pilotes de périphériques externes embarqués sont propriétaires. Cela rend le suivi énergétique tout à fait impossible en utilisant des schémas traditionnels tels que Powertrace. Par ailleurs, les auteurs contestent aussi la modélisation énergétique basée sur le temps d'utilisation d'un composant, montrant en effet que plusieurs composants périphériques peuvent avoir un schéma énergétique asynchrone. La solution proposée consiste à relier chaque composant matériel du système à un modèle énergétique sous forme de FSM [3] (machine à états finis) centré sur les appels systèmes décrivant son comportement. Le prototype est ainsi basé sur le suivi et l'enregistrement des appels systèmes et des routines exécutés par l'application. Les données collectées sont analysées et chaque routine se voit attribuer la quantité énergétique de l'état actuel du FSM. Dans le cas d'un comportement asynchrone, eProfMP attribue toujours la consommation énergétique à la dernière entité qui a déclenché ce comportement.

L'évaluation de eProfMP montre que l'instrumentation et l'enregistrement des traces ont une surcharge énergétique allant de 2 % à 15 %, ce qui peut donc entraîner une surcharge importante dans certain cas. De plus, la construction des modèles énergétiques FSM est réalisée manuellement, ce qui implique donc une quantité de travail considérable à effectuer pour chaque système matériel embarqué et pilote logiciel différent. Enfin, une précision de profilage satisfaisante n'est accessible qu'à condition d'accéder au code source interne de l'application, ce qui n'est souvent pas possible dans des cas réels.

Eon

Eon [SKG+07] n'est pas, à proprement dit, un profileur énergétique mais un environnement d'exécution et un langage dédié (DSL) ciblant des systèmes embarqués à fortes contraintes énergétiques. Les caractéristiques de Eon peuvent aider à un développement plus efficace grâce à l'augmentation du système cible avec des fonctions conscientes de la consommation énergétique courante. Le langage dédié proposé est basé sur un langage de coordination existant appelé *Flux* [BGK+06]. Ce langage est adapté aux systèmes orientés événement et permet à un développeur d'écrire les réponses pour chaque événement (externe ou interne) comme une séquence d'opérations (flux).

Afin d'ajouter des fonctionnalités de gestion de l'énergie, Eon étend ce DSL en y incorporant trois nouvelles notions :

3. FSM : Finite State Machine

— *les états énergétiques*, qui sont utilisés pour spécifier différents flux en fonction des état énergétiques courant du système ;
— *les timers adaptatifs*, qui permettent de définir un intervalle de valeurs pouvant être utilisé par les timers selon l'énergie encore disponible ;
— *les chemins énergétiques*, qui permettent de mettre en place de multiple chemins conditionnels dans l'application, où le plus approprié sera choisi lors du fonctionnement effectif du système par l'environnement d'exécution Eon.

L'évaluation d'Eon montre un impact positif sur la durée de vie du système. Ainsi, Eon est capable de choisir avec précision la meilleure politique énergétique à adopter. Une erreur moyenne de 10 % est à prendre en compte dans l'estimation de la consommation énergétique. De plus, la surcharge causée par la réévaluation de l'exécution de chaque flux est significatif. Enfin, la portabilité d'Eon sur différentes plates-formes embarquées peut être une tâche complexe.

Autres travaux

La majorité des travaux existants basés sur une approche de type logiciel ne se focalisent pas totalement sur le cas des petits systèmes embarqués. Cependant, ils proposent des méthodes avec une vaste gamme d'applications. Ces méthodes peuvent être utilisées ou plutôt adaptées pour le cas des petits objets connectés.

D'autre part, ces travaux sont inspirés en grande partie par une approche fondamentale, PowerScope[FS99]. PowerScopes fournit une estimation détaillée de la consommation énergétique d'un système mobile au niveau processus. Néanmoins, la faisabilité de cette approche nécessite des calculs et traitements supplémentaires de la part du système mobile cible. De plus, les développeurs doivent utiliser un ensemble d'API logiciels spécifiés par PowerScope afin de pouvoir estimer la consommation énergétique du système.

AppScope

Nous pouvons aussi citer AppScope [YKJ+12] qui cible les applications mobiles (à l'instar d'eProfMP). AppScope est implémenté en tant que module noyau (i.e. kernel) dans le système d'exploitation mobile Android. Il utilise une méthode dirigée par les événements pour surveiller l'utilisation par l'application des composants matériels du système, au niveau du noyau.

Lors de la détection d'événements, les statistiques d'utilisation requises par le modèle d'estimation sont collectées pour chaque composant matériel. Le modèle énergétique sous-jacent utilise ces statistiques afin d'estimer la consommation énergétique d'un processus Android de l'application mobile. L'évaluation d'AppScope sur plusieurs composants indique une estimation avec une bonne précision (erreur : $\pm$ 4.5%) de la consommation énergétique. La surcharge induite par le système d'estimation sur l'application en exécution est estimée à 35 mW en terme de consommation énergétique et à 2.1% en terme d'utilisation du processeur.

Eprof

Dans le même périmètre que eProfMP et AppScop, Eprof [SKZS12] est un autre profileur énergétique dédié au processeurs embarqués. La consommation énergétique synchrone est comptabilisé par l'approche en utilisant

un profilage énergétique statistique [MB06]. La solution capture une trace de la pile d'exécution lorsque les compteurs de performance matérielle du système déclenchent une interruption. Cette interruption correspond à un seuil de compteur atteint. Chaque déclenchement est lié à une quantité d'énergie pré-fixée, permettant ainsi l'association d'un bloc du code source avec une quantité correspondante de l'énergie consommée.

L'attribution de la consommation énergétique asynchrone est, quant à elle, plus complexe. À chaque utilisation d'un composant matériel du système mobile, l'approche enregistre une trace de la pile d'exécution actuelle ainsi que la requête émise au composant. Cependant une instrumentation de la structure de données de ces requêtes est nécessaire afin de réaliser cette trace. En ayant une vue complète sur le code source initiateur de la requête et le composant matériel ciblé, la comptabilisation de la consommation énergétique asynchrone devient possible lorsque le composant traite effectivement la requête.

L'évaluation d'Eprof montre une erreur d'estimation moyenne égale à 10% et une surcharge du temps d'exécution de 2,7%. Cependant, l'instrumentation du noyau exécuté peut avoir une incidence sur la stabilité du système. Enfin, un travail intensif doit être fait pour spécifier des modèles énergétique pour chaque composant périphérique embarqué.

ECOSystem

Nous pouvons aussi citer la solution ECOSystem [ZELV02] [ZELV03] [EL05], une approche logicielle basée sur un modèle nommé *currentcy model*. La principale caractéristique de ce modèle est l'utilisation d'une unité commune appelée *currentcy* pour la comptabilité de la consommation énergétique dans une variété de composants matériels et de tâches logicielles du système embarqué cible. L'unité *currentcy* est une abstraction pour représenter explicitement l'énergie en tant que ressource système, en précisant des objectifs liés à cette ressource et en capturant les interactions entre les composants du système consommateur en énergie.

L'évaluation du coût énergétique est basée sur un modèle en deux parties : la première partie résume la consommation énergétique de base du système cible et ses composants embarqués, tandis que la seconde décrit les états énergétiques actifs (i.e. lors de l'utilisation effective du composant). Le *framework* est évalué selon de multiples critères tels que la comptabilité de la consommation énergétique asynchrone ou encore la flexibilité de transposer et adapter ce *framework* pour supporter d'autres plates-formes matérielles.

2.5.3 Les approches basées sur la simulation

PowerTOSSIM

PowerTOSSIM [SHCW04] est un simulateur énergétique qui étend le simulateur initial TOSSIM [LLWC03]. Ce dernier est un environnement de simulation à évènements discrets pour les applications embarquées TinyOS [LMP+05] (un système d'exploitation pour les réseaux de capteurs). Grâce à l'utilisation de Tossim, PowerTOSSIM choisit d'éviter une simulation à grain fin (au niveau de l'instruction assembleur) lourde et coûteuse.

Fondamentalement, PowerTossim propose deux mécanismes distincts pour estimer la consommation énergétique du CPU et des autres composants matériels. Pour ces derniers (à l'exception du CPU à l'état actif), PowerTOSSIM instrumente la version simulée des pilotes logiciels des composants, en y insérant des triggers

qui sont enclenchés lorsqu'une transition d'état énergétique a lieu. Les triggers émettent un appel vers un module logiciel spécifique appelé *PowerState* qui enregistre toutes les transitions d'états énergétiques. En gardant cette trace, le simulateur est ainsi capable de suivre et d'estimer la consommation énergétique de chaque composant du système.

Afin d'analyser la consommation du processeur, PowerTOSSIM fait deux hypothèses : i) les systèmes embarqués utilisés dans les réseaux de capteurs sans fil embarquent des CPU simples, ii) la consommation énergétique "instantanée" est presque égale entre toutes les instructions CPU. Sur cette base, PowerTOSSIM instrumente le binaire de l'application embarquée pour y insérer un compteur gardant trace du nombre d'exécution de chaque bloc de base du code source. Ensuite, Le simulateur met en correspondance chaque bloc de base avec l'ensemble des instructions correspondantes obtenues en désassemblant le binaire de l'application. Enfin, PowerTOSSIM utilise la *datasheet* matérielle du système cible afin de récupérer la consommation énergétique de chaque instruction et calculer le nombre de cycle que dure chaque bloc de base du code source.

Malgré le temps de simulation optimisé et raisonnable atteint, PowerTOSSIM présente plusieurs inconvénients majeurs :

— La précision de l'estimation énergétique sur le CPU reste très versatile en raison de la surcharge du code simulé ;
— Il n'y a aucune prise en charge des mécanismes DVFS présents dans certains processeur/microcontrôleurs ;
— Il n'y a aucun modèle énergétique sous-jacent fournit par le simulateur, induisant donc la nécessité de mettre en œuvre un modèle spécifique pour chaque composant matériel ou alors d'en utiliser un déjà existant [DMN10].

eSimu

Présenté dans [FFF09] et [FFF07], eSimu est un simulateur énergétique qui diffère considérablement de PowerTOSSIM. eSimu modélise la consommation énergétique du système cible en tenant compte de la consommation énergétique par groupe d'instructions processeur, en plus d'une caractérisation énergétique de plusieurs périphériques externes. eSimu représente une approche à plusieurs étapes. Une première étape, similaire à PowerTOSSIM fournit une description précise de l'activité du système cible en mode simulation. La sortie générée est une représentation simple de toutes les instructions processeur exécutées avec leur nombre de cycles CPU,et de tous les événements des composants périphériques (i.e. les changements d'états énergétiques).

Les auteurs conçoivent également un modèle énergétique qui se veut générique, principalement centré sur la consommation énergétique du processeur (1).

$$\mathbf{E_{slot}} = \mathbf{E_{base}} + \mathbf{E_{CPU}} + \sum_{\mathbf{blocks}} \mathbf{E_{bl}} \tag{2.7}$$

Le modèle proposé est architectural, il représente le système cible comme un ensemble de blocs fonctionnels : CPU, mémoire et périphériques. E_{slot} représente la consommation énergétique durant un intervalle de temps précis. Ce dernier est choisi comme étant l'instruction exécutée en cours, donnant ainsi au modèle une sorte de granularité de niveau instruction. Le E_{base} représente la consommation énergétique du système quand celui-ci est dans l'état IDLE (*i.e.* état pendant lequel le système est inactif). La consommation énergétique de tous les blocs

architecturaux est exprimée en fonction de la surcharge énergétique par rapport à E_{base}. Pour les périphériques embarqués (i.e. $\sum_{blocks} E_{bl}$), les auteurs proposent, à l'instar des autres simulateurs, un modèle énergétique basé sur les états énergétiques que peuvent impliquer ces périphériques. Afin de calibrer le modèle énergétique, des micro-benchmarks sont effectués pour profiler les fonctionnalités du CPU et de certains composants périphériques [FFF06].

eSimu corrèle la trace d'exécution générée avec les données d'étalonnage du modèle, afin de fournir les estimations de consommation énergétique du code source de l'application. Les estimations sont affichées sous deux formes. Une première représentant un graphique composé de courbes énergétiques. Le second où les estimations sont résumées sous forme d'un arbre d'appel, où chaque nœud représente une fonction indexée par sa consommation énergétique.

AEON

AEON [LWG05] est un simulateur énergétique ciblant les réseaux de capteurs sans fil, qui reste assez similaire à PowerTOSSIM sur plusieurs points. Il propose un modèle énergétique basé sur la consommation en courant d'un nœud, ainsi qu'une exécution précise au cycle près du code applicatif réel. Le modèle énergétique est conçu afin de permettre la comparaison entre différentes approches d'économie d'énergie en terme d'efficacité énergétique. L'approche de simulation proposée consiste en trois étapes :

— Calibrer le modèle énergétique en mesurant la consommation en courant de tous les états énergétiques des composants matérielles intégrés au nœud ;
— Le modèle dérivé et résultant est implémenté dans un émulateur pour réseau de capteurs ;
— Le modèle est validé par des mesures via oscilloscope et des tests de durée de vie des batteries en exécutant des applications embarquées TinyOS.

Afin de réaliser ces étapes, AEON est implémenté sur AVRORA [TLP05], un émulateur pour nœud de réseau de capteur hautement évolutif montré comme étant plus performant que l'émulateur ATEMU [PBM+04]. Le modèle énergétique étend l'implémentation de chaque composant matériel dans AVRORA en surveillant leur comportement énergétique pendant l'émulation. Une étape de profilage énergétique est effectuée afin de permettre d'attribuer à chaque composant matériel et à chaque routine du code source leur contribution dans la consommation énergétique globale.

Cooja

Cooja [EÖV+09] [ÖED10] est un simulateur de réseaux de capteurs conçu pour les plates-formes embarquant Contiki-OS. COOJA exécute (i.e. simule) l'application Contiki sur chaque capteur avec une précision au cycle près et enregistre tous les états énergétiques des composants matériels. Plus précisément, chaque nœud du réseau est émulé par Cooja grâce à un émulateur intégré (principalement MSPSim pour les plates-formes MSP). Il en résulte un environnement où le réseau et le médium radio sont simulés, mais où les capteurs sont émulés, exécutant ainsi une application embarquée concrète.

Cooja n'est pas à proprement dit un profileur énergétique, mais il peut être utilisé pour déboguer différents bugs énergétique. De plus, Cooja est complété par un visualiseur nommée *Cooja Timeline* qui permet de tracer les activités des composants embarqués de chaque capteur avec une chronologie précise.

2.6 Discussion

Il est assez difficile de comparer les approches de profilage énergétique en raison des différents fondements sous-jacents de chaque solution. Chaque outil est basé sur des fonctionnalités particulières, dans le but de cibler différents objectifs spécifiques de profilage. Ainsi, selon l'approche utilisée, les axes de gestion de la consommation énergétique ciblés par un développeur embarqué diffèrent.

Néanmoins, afin d'obtenir une comparaison croisée efficace, nous considérons les critères précédemment décrits dans cet état de l'art. Le tableau 2.2 résume la pertinence de chaque solution de profilage énergétique selon de chaque critère. En essayant de classer chaque outil de profilage selon un critère spécifique, nous essayons d'extraire les tendances et les outils de profils les plus appropriés à l'objectif souhaité par le développeur.

Profileur	Estimation de la durée de vie	Gestion en ligne de la consommation énergétique	Granularité de l'estimation de la consommation énergétique	Assistance au développement efficace en énergie	Coût effectif	Coût lié à l'utilisation et le portage
ICount and Quanto	+	++	+++	+	++++	++
GoldCaps	+++	-	+	-	++++	++++
FlockLab	+	-	+	++	+	++
Energy Endoscope	+	-	++	++	+	+
Nemo	+	+++	+	+	++	++
PEEK	+	-	++	++++	++	+++
SPOT	+	+++	+	+	+++	++
Energy Bucket	++	-	+	+	++	+++
IAR I-jet debugger	+	-	+++	+++	+	++
Atmel debugger	+	-	++	+++	++	++
MAGEEC board	+	-	++	+	+++	+++
PowerTrace	+	++	+	+	NA	++
eProfMP	+	-	+++	++	NA	+
Eprof	+	-	+++	+	NA	+
AppScope	+	-	++	+	NA	+
Eon	++	++++	++	+++	NA	++
ECOSystem	+	++++	++	+	NA	+
PowerTOSSIM	+++	-	+++	+	NA	+
eSimu	++	-	++++	+++	NA	+
AEON	+++	-	+++	+	NA	+
Cooja	+	-	+	+++	NA	+

TABLE 2.2 – Comparaison des approches de profilage étudiées : la précision d'un outil pour un critère donné est indiqué par le nombre de "+" présent. Le symbole "-" indique une précision faible et "NA" signifie une non attribution pour ce critère.

Estimation de la durée de vie :
Dans le cas de la capacité d'*estimation de la durée de vie* de chaque outil de profilage, il semble assez difficile de positionner avec précision les approches. En effet, prédire ou estimer à l'avance la durée de vie en terme énergétique d'un système embarqué n'est certainement pas une tâche simple.

De l'étude faite sur toutes les approches, nous pouvons affirmer que chaque profileur énergétique est en mesure de prédire une estimation effective de la durée de vie d'un système embarqué. Cependant, seuls quelques outils rendent cette estimation possible en tant que fonction par défaut de ce dernier, sans aucun calcul supplémentaire du développeur. Nous pouvons citer les approches GoldCaps, PowerTOSSIM, AEON. La majorité de ces approches sont majoritairement de type simulateur énergétique et nécessitent un modèle de batterie pour fournir une estimation de la durée de vie. En revanche, les autres outils de profilage fournissent un profil localisé, qui n'est pas en capacité d'afficher directement une consommation énergétique globale du système embarqué ciblé.

Par ailleurs, indépendamment de la prédiction de la durée de vie faite lors de la phase de développement, de multiples perturbations et phénomènes ont une incidence sur la consommation énergétique réelle du système lorsqu'il est déployé dans son environnement final. Par conséquent, un écart important peut exister entre l'estimation produite et la consommation énergétique réelle prise lors du fonctionnement effective du système. Cette hypothèse est particulièrement essentielle à considérer lorsqu'il est question de profilage basé sur la simulation. Ce type de profilage est centré sur un modèle énergétique spécifique construit sur des conditions environnementales spécifiques.

Gestion en ligne de la consommation énergétique :
Les estimations de la durée de vie énergétique faites par les outils de profilage peuvent devenir imprécises ou même fausses après le déploiement effectif du système embarqué cible. Cela se produit principalement en raison de conditions environnementales variables et peu modélisables (i.e. bruit, interactions humaines, sources d'énergie, etc.). Pour ces situations, certains profileurs énergétiques tendent à offrir des fonctionnalités intégrées de gestion de la consommation énergétique en temps réel, après déploiement du système.

ECOSystem et Eon sont les meilleurs exemples puisqu'ils sont totalement construits autour de ce critère. Eon propose au développeur langage spécifique (DSL) pour exprimer divers chemins d'exécution en fonction de l'énergie résiduelle. ECOSystem quant à lui définit un concept unifié nommée *currentcy model* qui permet de distribuer l'énergie du système parmi les différentes tâches du système embarqué afin d'atteindre une durée de vie ciblé par le développeur. À un niveau plus bas, PowerTrace fournit au système embarqué hôte, par son instrumentation à faible coût, un moyen fonctionnel de connaître au moment de l'exécution une estimation de l'énergie consommée par les principaux composants embarqués du système.

En ce qui concerne les méthodes basées sur le matériel, Nemo et Spot, de manière similaire, donnent de par leur conception un moyen pour le système embarqué de mesurer sa propre consommation énergétique après déploiement. Quanto via ICount fournit un service comparable. De ce fait, il met en place des fonctionnalités d'un plus haut niveau d'abstraction pour régler et gérer la gestion de l'énergie du système embarqué, en mettant en avant un profilage à grain fin.

Par définition, les techniques de simulation n'identifient pas ce critère.

Granularité de l'estimation de la consommation énergétique :
Une bonne granularité dans l'analyse de la consommation énergétique est nécessaire dans de nombreuses situations. Par exemple, un développeur peut tracer et localiser directement le bloc de base du code source responsable d'un point chaud de consommation énergétique et déboguer de manière plus simple et efficace le problème.

À cette fin, les outils de profilage basés sur la simulation (mise à part Cooja) sont ceux qui proposent la granularité la plus fine, passant d'une granularité de bloc de base à une instruction unique. AEON est conçu sur un simulateur précis au cycle près. eSimu utilise un modèle énergétique de granularité d'instructions complexes. PowerTOSSIM effectue une conversion binaire pour récupérer la consommation énergétique d'un bloc de base.

Sur le plan logiciel, Eprof et eProfMP sont les plus performants. Ils proposent un profilage énergétique à un niveau "routine", à travers l'instrumentation des *syscalls* et du noyau, respectivement.

Enfin, nous remarquons que l'obtention d'un niveau de profilage raffiné n'est pas chose triviale en ce qui concerne les méthodes basées sur le matériel. La plupart de celles-ci fournissent au développeur un profilage de niveau "routine" simple. Généralement, ces méthodes utilisent une étiquette logique via une GPIO reliée à leur circuit de mesure matériel de consommation. Quanto et le débogueur IAR I-jet sont des exceptions. Le premier fournit un profilage énergétique multi-granularité en contrepartie d'un algorithme de calcul lourd. En revanche, le débogueur IAR se focalise entièrement sur sa sonde de débogage coûteuse et la disponibilité d'une unité de débogage ETM sur les cartes ARM ciblées pour atteindre un profilage de résolution "instruction". Malheureusement, ce dernier ne tient pas compte des consommation énergétique asynchrones existantes.

Assistance au développement efficace en énergie :
La granularité de profilage est souvent considérée et mise en évidence comme l'échelle de puissance d'une approche de profilage énergétique. Néanmoins, inonder le développeur avec une grande masse d'informations et le noyer sous des chiffres de consommation énergétique brute peut clairement complexifier le développement embarqué. Aussi, aider le développeur durant le profilage grâce à une utilisation facilitée de l'outil est un aspect important à prendre en compte pour une assistance complète au développement. Sur ce critère, il n'y a qu'une minorité d'outils étudiés qui fournissent une réelle aide au développeur ainsi qu'un retour formel et compact des résultats énergétiques.

Dans la catégorie des outils de type simulation, nous pouvons seulement citer eSimu et Cooja qui fournissent ces données de manière visuelles. En effet, Cooja est entièrement basé sur la visualisation et la gestion des nœuds de capteurs embarqués. Les solutions industrielles, IAR et ATMEL, augmentent directement leur IDE existant avec des fonctions de profilage énergétique.

La meilleure proposition reste le framework PEEK qui fournit également un SDK de développement entièrement centré sur la consommation énergétique et suggère au développeur certaines bonne pratiques énergétiques spécifiques, en mettant en opposition plusieurs configurations instantanées du code source.

Une approche différente mais toujours centrée sur le développeur est choisie par la méthode Eon. Celle-ci fournit au développeur un DSL complet pour lui permettre de gérer facilement les décisions énergétiques prises dans l'application embarquée en exécution après déploiement.

Coût :
Peu importe la façon dont une approche de profilage peut atteindre et accomplir chaque critère, son coût reste ce qui définit la faisabilité et le facteur de diffusion dans les secteurs industriels et de recherche.

Nous distinguons dans un premier temps, le coût effectif ou le prix, qui prend du sens concernant les approches matérielles mais aussi toute autre approche protégée par une licence propriétaire onéreuse à acquérir. Il n'y a pas de véritable courbe résumant le coût précis de chaque approche. Nous distinguons donc d'abord, les conceptions matérielles de mesure coûteuses qui sont basées sur des FPGA (Endoscope énergétique) ainsi que la JTAG Probe (IAR). Également, nous pouvons citer des circuit de mesure à faible coût basés sur de simple condensateur (i.e. ICount, GoldCaps).

Le deuxième aspect du coût est lié à la facilité de transfert de la solution. En effet, le temps passé à adapter la solution ou l'approche à ses propres besoins (e.g. une plate-forme embarquée cible différente) devient directement un coût effectif du développement, en particulier pour les acteurs industriels.

Les méthodes basées sur la simulation représentent certainement les pires candidats. Ces approches s'intègrent facilement dans un cycle de développement, néanmoins elles sont principalement basées sur des modèles énergétiques qui sont souvent difficiles à reproduire sur une plate-forme embarquée différente ou avec des attributs environnementaux de déploiement différents.

Les méthodes logicielles font un peu mieux. Néanmoins, la plupart de celles-ci instrumentent profondément le code source embarqué cible (e.g. PowerTrace, eProfMP) ou proposent une base logicielle complexe et lourde (Eon, ECOSystem). Dans tous les cas, cela pourrait représenter une barrière forte et dure à outrepasser pour adapter et gérer une solution logicielle sur une plate-forme embarquée cible d'une architecture différente.

Enfin, en étant principalement agnostique du logiciel, les propositions faites par les profileurs de type matériel semblent généralement donner et fournir la meilleure flexibilité de portage.

2.7 Conclusion

La capacité à profiler la consommation énergétique d'une application embarquée représente l'une des clés primordiale afin de passer outre les barrières existantes dans l'Internet des objets, aujourd'hui.

Nous avons présenté dans ce chapitre d'état de l'art une vue large de la consommation énergétique au sein d'un objet connecté. Nous avons décrit les connaissances nécessaires pour appréhender le fonctionnement de ce phénomène physique et d'en mesurer et quantifier l'ampleur. Nous avons exposé en détail les approches existantes qui apportent des idées innovantes et des avancées prometteuses dans le domaine du profilage énergétique des applications embarquées. Ces approches ont été classifiées selon trois catégories et ont été évaluées selon cinq critères distincts. En étudiant ces approches et en mettant en évidence leurs avantages et leurs inconvénients, nous avons pu montrer qu'il n'existe pas réellement de méthode répondant à tous les critères. Ainsi, un développeur doit choisir la bonne catégorie de profilage et la technique qui lui convient le mieux, en fonction du contexte et des impératifs du projet.

De par le contexte fortement industriel de notre travail de thèse, les impératifs à prendre en compte sont :

— une bonne précision de la mesure énergétique, avec donc un faible taux d'erreur ;
— une grande hétérogénéité dans les systèmes embarqués utilisés. Ce qui implique un portage simple et non onéreux de la solution de profilage ;

— des développeurs non sensibilisés à la problématique énergétique ayant besoin d'être guidés sur les questions énergétiques durant le développement.

Afin de répondre au mieux à ce cahier des charges industriel et recherche, nous plaidons pour l'utilisation d'une approche de profilage basée sur une mesure matérielle. Notre but est de profiter des avantages d'une telle approche tout en atténuant les défauts. Nous explorons dans le prochain chapitre les atouts possible de ce choix et de cette voie.

«««< HEAD ======= »»»> 468305393ff4e54ce221af07efd6845f6a2f2d09

Chapitre 3

Simulation Vs Réalité

- Tu bluffes, Martoni.
- Moi je suis de l'avis de Bialès, il bluffe. On vote ? Moi je vote et je dis il bluffe.
Gérard Darmon, Alain Chabat et Dominique Farrugia, La Cité de la peur (1994)

3.1 Introduction et Motivation

L'état de l'art établi dans la section précédente nous a permis de nous rendre compte des différentes techniques de profilage énergétique existantes. Nous avons défini et décrit la complexité existante lorsqu'il s'agit de comparer ces techniques et de ce fait en choisir la plus pertinente. Néanmoins, dans l'optique de fournir au développeur une méthode précise et généralisable de par les différentes plates-formes embarquées existantes, nous écartons de notre travail de thèse les méthodes basées sur la simulation afin d'être au plus proche du "*monde réel*".

Afin d'étayer notre propos, nous menons et décrivons dans ce chapitre une expérimentation de mise en opposition d'une méthode de mesure énergétique basés sur la simulation et d'une méthode logicielle. Notre objectif est de montrer et modéliser expérimentalement le gap qui existe entre les chiffres fournis par les deux techniques de mesure. À travers cette différence, nous appuyons le fait qu'une méthode expérimentale permet au développeur d'acquérir une vraie compréhension de la consommation énergétique de son application embarquée.

Dans la suite de ce chapitre, nous présentons l'expérimentation entreprise. Celle-ci met en œuvre un réseau sans fil multi-sauts comportant sur le nœud le plus éloigné un serveur web embarqué. Un tel cas d'usage permet d'imager un scénario réel du monde embarqué et de l'Internet des Objets d'aujourd'hui.

3.2 Un cas d'usage réel : réseau multi-sauts et serveur web embarqué

L'expérimentation de notre hypothèse précédente passe donc par la mise en place d'un cas d'usage réel et la mesure de la consommation énergétique des systèmes le composant. À cet effet nous mettons en place un réseau sans-fil multi-saut minimaliste dans la lignée des réseaux de la littérature académique. En outre, afin d'avoir une application ressemblant à ce que nous pouvons rencontrer aujourd'hui, nous plaçons un serveur web embarqué sur l'extrémité du réseau, réalisant ainsi une des idée du Web des Objets (ang. *Web of Things*). Celui-ci va plus loin que le paradigme Internet des objets en ajoutant l'interopérabilité Internet à la couche applicative. Il encourage l'utilisation de services Web afin d'intégrer pleinement les objets connectés aux systèmes informatiques existants et ainsi leur permettre d'évoluer au fur et à mesure que le cas d'utilisation se développe.

««««< HEAD Afin de mettre en place ce réseau de capteurs, nous utilisons le système d'exploitation pour petit objet communicant, Contiki OS.

Contiki OS [DGV04] est défini comme un système d'exploitation open source pour l'Internet des objets. Il est destiné au petits objets connectés contraint en puissance, en mémoire et aussi en énergie. Malgré la faible empreinte mémoire et énergétique de Contiki OS, il supporte complètement les standards d'Internet. Il offre ainsi une compatibilité en IPv4 et IPv6 pour la couche réseaux ainsi qu'un support de différents protocoles applicatifs dont HTTP et CoAP. Il reste l'un des systèmes d'exploitation les plus utilisé dans le monde académique (et aussi industriel) pour les réseaux de capteurs de sans fils. ======= Afin de mettre en place ce réseau de capteurs, nous utilisons le système d'exploitation pour petit objet communicant, Contiki OS [DGV04]. Il est défini comme un système d'exploitation open source pour l'Internet des objets. Il est destiné au petits objets connectés contraint en puissance, en mémoire et aussi en énergie. Malgré la faible empreinte mémoire et énergétique, il supporte complètement les standards d'Internet. Ainsi, Contiki OS offre une compatibilité en IPv4 et IPv6 pour la couche réseaux ainsi qu'un support de différents protocoles applicatifs dont HTTP et CoAP. Il reste l'un des systèmes d'exploitation les plus utilisé dans le monde académique (et aussi industriel) pour les réseaux de capteurs de sans fils. »»»> dd6e1c9f6f697f983e314dedce5a84b26e420687

Afin de rendre effective la compatibilité avec les standard d'Internet, notamment lors de son application à un réseau de capteurs multi-sauts, Contiki OS met en place plusieurs optimisations énergétiques au sein de sa pile de communication. La compréhension de ces optimisations étant primordiales pour l'analyse des résultats de l'expérimentation. Nous donnons, dans la suite de cette section, une description qualitative de ces optimisations à travers la pile de protocoles de communication.

3.2.1 Les optimisations énergétiques au niveau de la couche MAC

La couche MAC radio représente la couche logicielle la plus basse dans une pile de communication. En outre, elle dirige l'activité réelle et effective du module radio de l'objet communicant dans le but d'effectuer les opérations nécessaires (écoute, transmission, etc.). Dans l'exemple de Contiki OS, la majorité des modules radio supportés fonctionnent via la norme radio 802.15.4 [Soc03]. Le module radio étant certainement l'un des composants les plus énergivore dans un objet communicant [Dun11], il est communément admis et mis en œuvre d'éteindre ce module le plus souvent possible afin d'économiser une quantité maximale d'énergie [PHC04]. Les protocoles MAC (i.e. couche MAC) pour objets communicants sont tous conçus à cet effet. Ils pilotent le module

radio pour que ce dernier se réveille uniquement périodiquement pour inspecter si un paquet radio lui est destiné. Ces protocoles sont d'ailleurs plus communément nommés protocoles RDC (ang. *Radio Duty Cycle*).

Afin d'être - plus au moins - sûr de pouvoir communiquer avec un objet voisin lors de son réveil, un objet transmetteur utilise souvent un mécanisme d'envoi de paquets dit "par sondes", en rafales.

Plusieurs protocoles MAC efficaces en énergie ont déjà été proposés dans la littérature [PHC04] [BYAH06] [ML08]. ContikiMAC [Dun11] est certainement l'un des plus populaires d'entre eux. À l'instar des autres protocoles, il utilise un protocole de réveil périodique ainsi qu'un mécanisme de transmission de sondes qui sont ici de simples copies du paquet réel à envoyer. ContikiMAC met en œuvre trois optimisations importantes pour améliorer l'efficacité énergétique :

— il utilise un mécanisme précis de réveil du module radio, reposant sur la librairie temps réel intégrée dans le système d'exploitation Contiki OS ;
— il implémente un mécanisme de mise en veille rapide du module radio, en permettant à ce dernier de différencier très rapidement entre un vrai paquet radio en cours de réception et du bruit ;
— il introduit aussi un mécanisme de synchronisation de phases permettant d'enregistrer dans un cache local, une liste de phase de réveil de chaque objet voisin. Ainsi, un objet voulant transmettre à un voisin et connaissant le moment exact de réveil de ce dernier, pourra débuter sa communication juste avant le moment prévu où ce voisin se réveillera. la consommation énergétique préjudiciable à l'envoi en rafales des paquets sondes s'en trouve ainsi amoindrit.

Nous illustrons en Figure 3.1 le résultat d'une expérience visant à montrer l'impact de la fréquence du protocole RDC de ContikiMAC sur la latence des transmissions. Nous avons effectué une requête *Ping* sur un objet communicant situé trois sauts plus loin, en utilisant ContikiMAC avec des fréquences de 8 Hz, 16 Hz et aussi sans utiliser du tout de protocole RDC (i.e. radio toujours allumée). Comme nous pouvions l'attendre, le choix du protocole RDC possède un grand impact sur la latence de bout en bout entre deux nœuds du réseau. Ainsi, nous remarquons facilement que la non utilisation d'un protocole RDC réduit la latence d'un facteur cinq en comparaison avec l'utilisation de ContikiMac à 8 Hz. Néanmoins cela vient avec l'inconvénient d'une radio active plus longtemps, impactant de manière négative la consommation énergétique de l'objet. La consommation énergétique représentant la métrique principale étudiée dans nos expérimentations, nous décidons d'utiliser une ContikiMac à 8 Hz afin de privilégier l'efficacité énergétique.

3.2.2 Les optimisations énergétiques au niveau de la couche IP

Après avoir discuté du sujet des protocoles MAC embarqués efficaces en énergie, nous passons au niveau supérieur avec la couche réseau du modèle OSI. Les protocoles Internet traditionnels à l'image d'IP (ang. *Internet Protocol*) n'ont pas été conçus, au tout début, avec des critères ou des objectifs d'efficacité énergétique. Néanmoins, avec l'engouement généré par l'Internet des Objets, la possibilité d'utiliser ces protocoles pour connecter des objets communicants contraints en énergie a été étudiée et prouvée par de multiples travaux [HC08] [Wes06].

Parmi ces travaux, uIP et lwIP [Dun03] proposés par Adam Dunkels représentent les premiers résultats conséquents tirant avantage de l'utilisation du protocole IP sur des médiums radio basse consommation énergétique pour objets communicants. Ces deux couches IP proposent de multiples approches pour limiter la

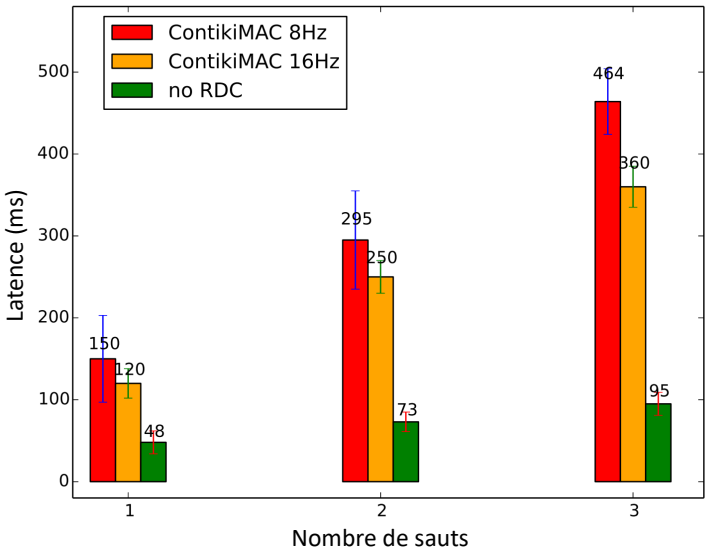


FIGURE 3.1 – Expérimentation sur la latence d'un *Ping* sur un réseau à trois sauts selon différents réglages de fréquence de la ContikiMAC.

surconsommation énergétique engendrée par l'utilisation du protocole IP. L'une des plus fortes surconsommation est dû à la lourdeur des entêtes IP par rapport à la faible bande passante du protocole radio 802.15.4. Cette problématique a été résolue en utilisant des techniques de compression d'en-tête. Enfin, uIP et lwIP proposent aussi un mécanisme léger de routage IP, qui permet d'atténuer la charge sur les objets hautement contraints d'un réseau.

Plus récemment, avec l'avènement du protocole IPv6, l'IETF (Internet Engineering Task Force) a proposé 6LoWPAN [Mul07] comme nouveau protocole réseau efficace en énergie. 6LowPan rend ainsi possible le passage de larges paquets IPv6 par un médium radio contraint à l'instar de celui définit par la norme IEEE 802.15.4. Plusieurs schémas de compression ont été définis et mis en œuvre ciblant plusieurs ratios et temps de compressions. Par ailleurs, IPv6 est caractérisé par une MTU (Maximum Transmission Unit) très large de 1280 octets. De ce fait, plusieurs mécanismes de fragmentation de paquets au niveau réseau ont été mis en place pour supporter, d'une manière optimale (en termes énergétique et temporel) cette MTU. Plusieurs implémentations de ce protocole existent, nous pouvons citer SICSlowpan (conçu pour Contiki OS) [Dun08] et b6loWPAN ou blip (conçu pour TinyOS) [GMS14]. Ces derniers représentent les travaux les plus matures en vue du support et de l'implémentation de 6LoWPAN. Ils incorporent ainsi les optimisations précédemment décrites en plus d'un mécanisme léger d'auto-assignation d'adresse IPv6 à l'intérieur d'un réseau d'objets.

3.2.3 Les optimisations énergétiques au niveau de la couche TCP

Lors de communications IP entre objets communicants, le protocole UDP est souvent préféré du fait de son faible surcoût. Néanmoins, afin d'être compatible avec des protocoles historiques et traditionnels applicatifs

tels que HTTP ou SSH, ainsi que pour correspondre aux critères de divers types de cas d'utilisation sensibles (médical, militaire, etc.), une garantie de fiabilité est nécessaire dans la délivrance des paquets émis.

Afin d'atteindre ce but, l'adaptation du protocoles TCP (ang. *Transmission Control Protocol*) a concentré de nombreux efforts ces dernières année. Dans [DVA04], Adam Dunkels et al. présentent un schéma distribué de cache TCP afin de réduire le nombre de retransmissions de bout en bout. Dans ce schéma, chaque objet du réseau maintient un cache pouvant accueillir un seul et unique segment TCP, permettant ainsi une retransmission direct en cas de perte de celui-ci. Une autre approche présentée dans [Bra89], tente de réduire le nombre d'acquittement (ACK) en tirant avantage du mécanisme d'acquittement retardé implémentée dans un grand nombre de piles TCP/IP traditionnelles. Sur le même thème, dans [AJ03], les auteurs étudient l'impact des acquittements retardés sur l'ensemble d'un réseau d'objet, et ils proposent un mécanisme dynamique de retardement d'acquittements afin d'améliorer le rendement énergétique du réseau.

3.2.4 Impact des éléments externes sur la consommation énergétique

Les optimisations énergétiques citées précédemment sont mises en œuvre par Contiki OS afin de réduire l'empreinte énergétique des communications sans fils. Cela est d'autant plus vrai lorsque ces communications sont faites à travers les standards d'Internet (IP, TCP, UDP, HTTP, ...). En outre, ces optimisations sur la pile de communication permettent aussi de limiter l'impact de phénomènes perturbateurs sur système. Nous pensons ici, en priorité, au bruit radio environnant qui génère des interférences pouvant perturber les communications sans fils. D'une importance moindre, des dérives physiques des horloges cadençant le système peuvent induire une désynchronisation temporelle entre les objets communicants, augmentant de ce fait la consommation énergétique globale.

Nous pouvons citer les travaux de Victor et al. [TIPV$^+$16] qui décrivent l'impact de la congestion des fréquences de communications sur la qualité et consommation énergétique des transmissions et réceptions de paquets au niveau de la couche MAC. Dans [HSL$^+$16] des travaux similaires sont décrits dans l'objectif de limiter cet impact.

Tous ces travaux se concentrent sur l'influence des interférences sur la plus basse des couches réseaux, la couche radio/MAC. À l'opposé, le travail décrit dans ce chapitre a pour but de modéliser l'écart existant entre l'estimation de la consommation énergétique dans le monde réel et dans un environnement simulé. Nous pensons qu'au vu de l'utilisation actuelle des objets connectés, les principes de l'internet des objets (IP, UDP, TCP) et du web des objets (HTTP) doivent être pris en comptes.

De ce fait, nous traitons et prenons en compte la consommation énergétique dans un système connecté complexe. Un système où la consommation énergétique est la conséquence finale du fonctionnement global de celui-ci induit par un besoin applicatif. Autrement dit, l'application qui dirige les taches fonctionnelles est la cause du coût énergétique. Cette cause se diffuse dans les couches logicielles, des plus abstraites vers les plus concrètes. Enfin, cette diffusion peut mettre en exergue, lors de la présence d'éléments et interférences externes, des "bugs" énergétiques augmentant la consommation énergétique attendue.

3.3 Protocole expérimental

Notre objectif dans ce chapitre est de déterminer et de quantifier l'écart existant entre l'estimation énergétique par le biais de simulateurs énergétiques d'une estimation faite dans le monde réel. Afin de mettre en évidence les différences existantes, nous avons décidé de mener l'expérimentation en utilisant un cas d'usage réel basé sur un réseau multi-sauts sans fil incluant un serveur web HTTP embarqué sur le nœud final du réseau.

En utilisant deux techniques de mesure énergétique, nous voulons quantifier les différences existantes entre simulation et monde réel. De par les constatations de ces expérimentations, nous pouvons ainsi mettre en évidence les faiblesses d'une estimation dans un environnement simulé (i.e. comprendre aussi le modèle de simulation). Nous présentons dans cette section la méthodologie utilisée pour arriver à cet objectif.

3.3.1 Méthodologie et banc de test expérimental

Nous effectuons nos expérimentations sur des plates-formes Tmote Sky [mot06]. Elles embarquent un micro-contrôleur 16 bit et un module radio CC2420 [Ins17] basse consommation fonctionnant selon la norme IEEE 802.15.4 [Soc03]. Toutes les plates-formes embarquent le système d'exploitation pour systèmes embarqués précédemment décrit Contiki OS.

Nous basons notre expérimentation sur des cas pouvant être réellement rencontrés dans la vie de tous les jours. Pour cela, nous mettons au point le banc de test schématisé en Figure 3.2. Ce banc consiste en un mini réseau sans fil de trois sauts réseaux. Les routes réseaux sont fixées statiquement afin d'éviter de mesurer involontairement les effets énergétiques causés par un protocole de routage dynamique. Trois plates-formes Tmote Sky composent et font entièrement partie du réseau. La troisième plate-forme, celle située le plus loin du routeur de bordure, embarque un serveur web HTTP qui aura comme fonction de répondre aux requêtes émanant de l'extérieur du réseau. Ces requêtes sont routées par les plates-formes intermédiaires du réseau multi-sauts. Une quatrième plate-forme joue le rôle de routeur de bordure IPv4/IPv6 et est utilisée pour connecter le réseau sans-fil à un PC fonctionnant sur Linux. Cette plate-forme utilise le protocole SLIP (Serial Line IP) [Rom88] pour transmettre et recevoir les paquets IP radios vers/depuis le PC.

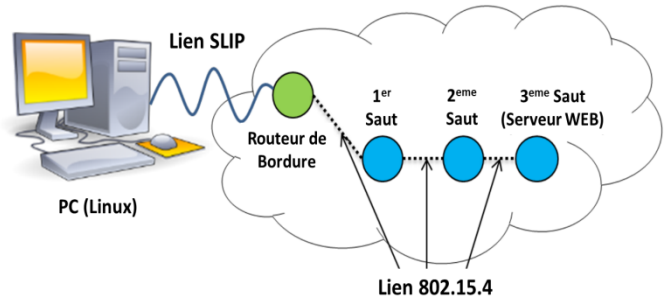


FIGURE 3.2 – Banc de test expérimental consistant en un réseau de capteurs sans fils.

Les modules radios de toutes les plates-formes utilisées sont configurés pour utiliser le canal de communication 15 de la norme 802.15.4. Celui-ci est connu pour être le canal le moins sensible aux interférences WiFi

environnantes. Les plates-formes sont déployées directement dans les locaux de nos laboratoires, un lieu où une activité humaine journalière est présente.

Afin de mesurer la consommation énergétique du serveur HTTP embarqué sur la troisième plate-forme (i.e. nœud), nous effectuons plusieurs expérimentations en utilisant notre banc de test. Plus précisément, le PC situé à l'extérieur du réseau initie des requêtes HTTP vers le serveur web embarqué localisé sur le dernier nœud. Ces requêtes demandent du serveur une charge effective allant de 1 à 1040 octets (i.e. 1040 octets représente la charge maximal pouvant être demandée en IPv6/6LoWPAN dans les limites des capacités de nos plates-formes) Les deux nœuds restants du réseau effectuent la transition/routage des requêtes et des réponses. Les requêtes sont initiées de manière individuelle et disjointe entre elles. Enfin nous utilisons la commande Linux CURL afin d'initier les requêtes HTTP du PC.

3.3.2 Configuration du banc de test

Nous effectuons nos expérimentations sous deux configurations de la pile de communication. Celles-ci sont les piles de communications uIPv4 et uIPv6 implémentées dans Contiki OS.

Pour uIP4, nous utilisons une MSS (ang. *Maximum Segment Size*) de 48 octets, définie par défaut au cœur de Contiki OS. Dans le cas d'uIPv6 la MSS utilisée s'élève à 1220 octets, activant de ce fait le mécanisme de fragmentation de paquets réseaux implémenté dans cette pile. Enfin, aucune optimisation TCP n'est incluse ou activée dans la version des piles de communication que nous utilisons. De ce fait, chaque segment TCP nécessite un acquittement. Néanmoins, les trois objets communicants du réseau implémentent le protocole MAC - efficace en énergie - ContikiMAC avec une fréquence de réveil de 8 Hz. Cette fréquence est choisie afin de se mettre dans un scénario où l'efficacité énergétique représente l'aspect primordial à optimiser en contrepartie de la latence des transferts.

3.3.3 Environnements de test

Nous effectuons nos tests sous deux environnements distincts : dans le premier, nous déployons notre banc de test dans un environnement réel représenté par nos locaux de laboratoire. Ces locaux à l'instar de tant d'autres, sont équipés de point d'accès WiFi et fréquentés par un certain nombre de personnes. De ce fait la probabilité de présence d'ondes WiFi ou Bluetooth (fonctionnant eux aussi sur les mêmes fréquences) est non négligeable.

Pour le deuxième environnement, nous utilisons le simulateur d'application Contiki OS, nommé Cooja [EÖV+09] [ÖED10]. Ce dernier offre un environnement d'expérimentation idéal, net de toutes interférences externes ou internes.

Afin de caractériser la consommation énergétique de chaque objet lors du traitement d'une requête/réponse HTTP, nous utilisons le profileur énergétique précédemment décrit dans le chapitre *état de l'art*, Powertrace [DEFT11]. En particulier, chaque objet sauvegarde deux estampilles temporelles (i.e. *timestamp*) Powertrace, l'une au début et l'autre à la fin de la requête HTTP, lors des détections respectives des drapeaux TCP "SYN" et le dernier "FIN-ACK". Ainsi pour chaque requête HTTP nous obtenons la consommation énergétique de chaque nœud lors du traitement de cette dernière.

Les résultats obtenus sont moyennés sur 50 exécutions afin de s'affranchir d'éventuelles perturbations ponctuelles. Sur chaque graphique, les bâtonnets gris représentent les écarts-type de ces 50 exécutions.

3.4 Résultats expérimentaux

Dans cette section, nous décrivons et illustrons les résultats de nos expérimentations. Nous présenterons ainsi respectivement les résultats obtenus lors des expérimentations en environnement de simulation et en environnement réel.

3.4.1 Simulation

Dans cette première partie, nous utilisons le simulateur Cooja afin de se mettre dans un environnement de simulation idéal tel que voulu par la conception de ce dernier. Nous effectuons grâce à cette configuration l'expérience décrite précédemment. Les figures 3.3 et 3.4 montrent respectivement la consommation énergétique des trois objets connectés en fonction de la charge utile en configuration IPv4 et configuration IPv6. Les légendes de ces graphiques peuvent être expliquées comme suit : le troisième saut illustré en vert représente le serveur web. Les premier et deuxième sauts illustrés respectivement en rouge et bleu correspondent aux routeurs à l'intérieur du réseau.

Observations générales

En premier lieu, nous pouvons apercevoir sur les deux expériences, une différence entre la consommation des trois objets. Ainsi, les deux nœud routeurs (premier et deuxième sauts) consomment plus d'énergie que l'objet serveur, du fait de leurs communications dans les deux sens de la route réseau. De plus, le premier routeur est caractérisé par une consommation énergétique moindre. Cela est explicable car le routeur de bordure, voisin de ce dernier, a son module radio qui est toujours actif. De ce fait, le premier routeur, contrairement au deuxième, ne perd pas d'énergie dans la transmission de plusieurs paquets sondes, en attente du réveil du routeur de bordure.

Observations et analyses : IPv4

Pour l'expérimentation en IPv4 (Figure 3.3), nous pouvons noter une incrémentation en escalier de la consommation énergétique de tous les nœuds à mesure que le nombre de paquets augmentent (i.e. dans notre configuration, tous les 48 octets). De plus, nous remarquons que l'écart-type symbolisant la variation de la consommation énergétique sur toutes les itérations est très faible (~1,5 mJ). Par ailleurs, il est complètement indépendant de la quantité de donnée transmise.

La consommation énergétique étant très stable et directement reliée au nombre de paquets dans la réponse HTTP, il est aisé de construire le modèle énergétique sous-jacent. On note E_{v4_i} la consommation énergétique en mJ dans la configuration IPv4 du i-ème objet sur la route. La variable "x" représente dans toutes les équations, la taille de la charge utile de la réponse HTTP en octet. Nous donnons comme exemple, l'équation énergétique du premier objet de la route (équation (3.1)). Concernant les deux autres objets restants, nous trouvons par rapport aux paramètres d'équations : $\alpha_2 = 3.5$, $\beta_2 = 22.7$, $\alpha_3 = 2$, $\beta_3 = 12$ et std_v4 1.5.

$$E_{v4_1} = \alpha_1 \lceil \frac{x}{48} \rceil + \beta_1 = 3.5 \lceil \frac{x}{48} \rceil + 19 \tag{3.1}$$

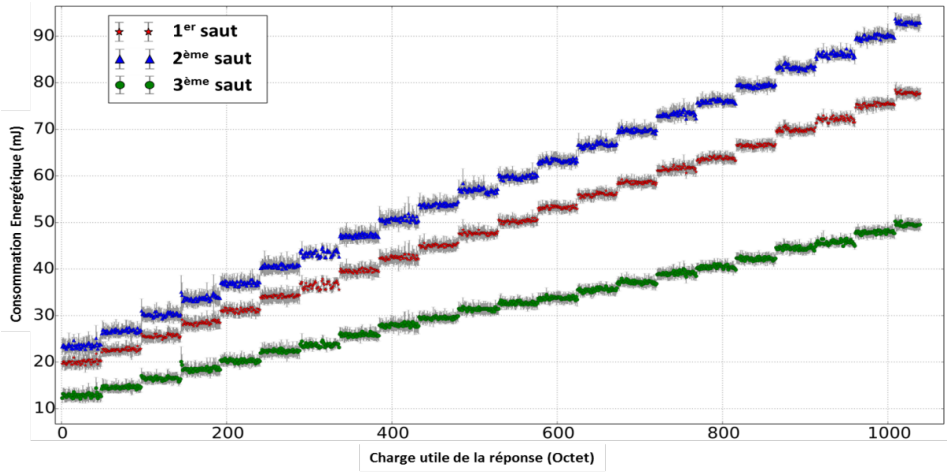


FIGURE 3.3 – Consommation énergétique des trois objets communicants du réseaux en environnement simulé lors de l'utilisation d'IPv4.

Observations et analyses : IPv6

Dans l'expérimentation IPv6 (Figure 3.4), la consommation énergétique n'est plus vraiment liée au nombre de paquets transitant sur la route, mais plus directement au nombre d'octets. La consommation est nettement moins grande en termes de valeur globale et d'incrémentation en fonction de la taille de la charge utile. Ceci est dû au fait qu'IPv6 permet d'utiliser une large MSS équivalent à 1220 octets. L'utilisation du mécanisme de fragmentation, implémenté dans la pile IPv6-6LoWPAN et décrit dans la RFC de la couche d'adaptation 6LoWPAN, donne la possibilité au serveur web embarqué de délivrer tout le contenu demandé en un seul et unique paquet IP. Enfin, l'écart-type reste aussi dans ce cas, assez faible et constant (~2,5 mJ).

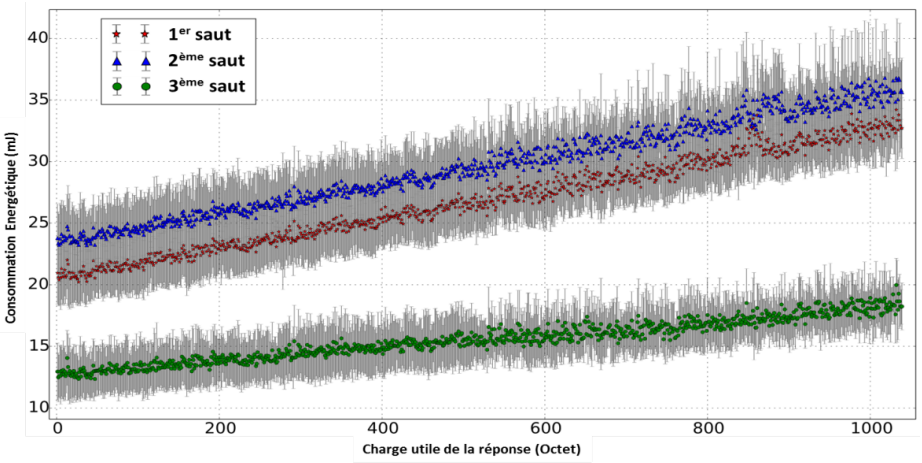


FIGURE 3.4 – Consommation énergétique des trois objets communicants du réseaux en environnement simulé lors de l'utilisation d'IPv6.

À l'instar de l'expérimentation précédente nous pouvons aisément modéliser la consommation énergétique sous forme d'une application linéaire. Nous notons E_{v6_i} la consommation énergétique en mJ, dans une configuration IPv6 du i-ème objet sur la route. Nous donnons comme exemple, l'équation énergétique du premier objet (équation 3.2) et nous trouvons concernant les deux autres objets restants : $\gamma_2 = 0.012$, $\varphi_2 = 23.2$, $\gamma_3 = 0.006$, $\varphi_3 = 12.7$ et std_v6 2.5.

$$E_{v6_1} = \gamma_1 x + \varphi_1 = 0.010x + 20.3 \tag{3.2}$$

Synthèse

En synthèse de cette première partie d'expérimentation, nous pouvons conclure dans un premier temps que la consommation énergétique sous un environnement simulé est relativement simple à modéliser. Nous avons vu que celle-ci suit un modèle linéaire auquel est associé une erreur (i.e. écart-type) très faible. Autrement dit, la consommation énergétique pour une requête donnée, reste stable quelque soit le moment où celle-ci est faite. Dans la suite de cette section, nous démontrons que cette affirmation n'est pas applicable lors de la mesure de la consommation énergétique pour un environnement réel.

3.4.2 Conditions réelles

Afin de montrer l'impact d'un environnement réel de déploiement sur la consommation énergétique d'un réseau d'objets communicants, nous réalisons les mêmes expériences que celles décrites précédemment en utilisant de véritables plates-formes embarquées Tmote Sky déployées dans nos bureaux. Les figures 3.5 et 3.7 décrivent respectivement la consommation énergétique des trois objets sur la route, en fonction de la taille de la charge utile dans les configurations IPv4 et IPv6.

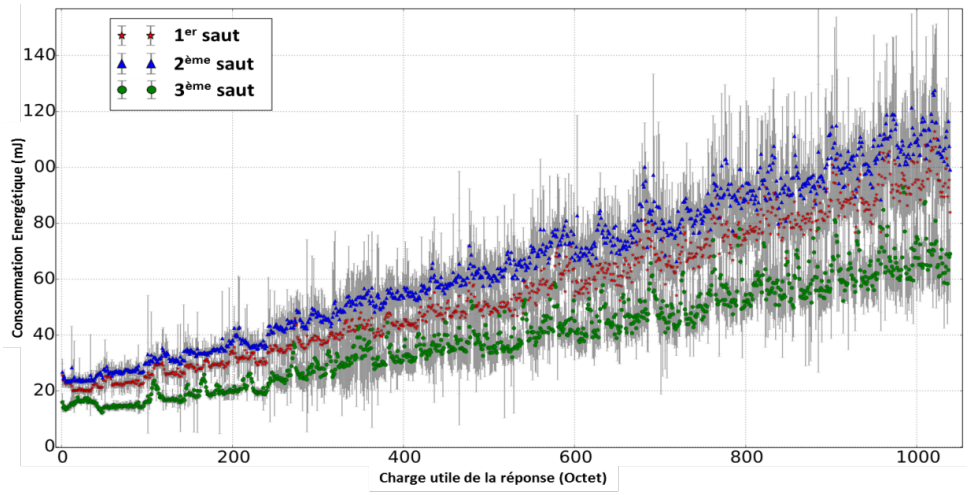


FIGURE 3.5 – Consommation énergétique des trois objets communicants du réseaux en environnement réel lors de l'utilisation d'IPv4.

Observations et analyses : IPv4

Nous nous focalisons dans un premier temps sur le cas de la configuration IPv4 en environnement réel. Nous pouvons remarquer que la consommation énergétique, contrairement au cas simulé, est ici directement liée à la taille de la charge utile transmise plutôt qu'au nombre de paquets. Cette consommation ainsi que l'écart-type associé sont aussi largement supérieure en comparaison du cas simulé, suivant une incrémentation non constante. De plus, nous observons une augmentation et baisse périodique en forme "pic" de la consommation énergétique sur les trois nœuds. Cette augmentation et baisse brutale se produisent lors de la transmission d'un trame ayant une quantité de donnée proche de la limite de contenance d'une trame radio 802.15.4 (i.e. 127 octets avec en-têtes comprises, ou 48 octets en terme de données utiles). Ce phénomène particulier est totalement absent dans le cas simulé présenté précédemment.

Sans définir "pour l'instant" l'écart-type, nous pouvons approximer la consommation énergétique par une fonction linéaire. Nous donnons en guise d'exemple l'équation du premier objet de la route (équation (3.3)). Pour les objets restants, nous trouvons : $\lambda_2 = 0.084$, $\psi_2 = 23.8$, $\lambda_3 = 0.049$, $\psi_3 = 13.6$ et std_v4

$$E_{v4_1} = \lambda_1 x + \psi_1 = 0.073x + 21 \pm std_v4 \tag{3.3}$$

Observations écart-type

À cause de l'écart-type élevé, le pouvoir prédictif potentiel de notre modèle linéaire est loin d'être assez probant en comparaison avec celui donné dans le cas simulé. La figure 3.6 représente l'écart-type de la consommation énergétique sur le deuxième nœud de la route du réseau d'objet. Nous illustrons ce nœud en particulier, car il représente parmi les trois nœuds de la route, le cas le plus souvent rencontré dans un réseau d'objets. Autrement dit, celui d'un nœud routeur faisant transiter les requêtes/paquets entre deux autres nœuds distants.

Nous remarquons que l'écart-type augmente d'une manière non stable, en fonction de l'augmentation de la taille de la charge utile demandée, pour atteindre une valeur haute équivalente à 36 mJ lors d'une requête demandant une charge effective de 1000 octets. De surcroît, l'augmentation de la dispersion des valeurs est elle aussi de plus en plus importante. Enfin, nous pouvons constater que l'écart-type peut d'une certaine manière se décomposé en deux ensembles mis en évidence par les droites $f1_{std_2}$ and $f2_{std_2}$ sur la figure 3.6.

Nous prenons donc en compte ce comportement pour approximer l'écart-type de la consommation énergétique. Nous notons Std_{v4_2} l'écart type en mJ du deuxième objet de la route ((3.4)).

$$Std_{v4_2} = \begin{cases} f1_{std_2} = \eta_a x + \mu_a = 0.01x + 1 \\ f2_{std_2} = \eta_b x + \mu_b = 0.028x + 1.6 \end{cases} \tag{3.4}$$

Synthèse : IPv4

En résumé, le développement et le déploiement dans un environnement réel, à l'opposé de celui simulé, requiert de la part du développeur d'applications embarquées de faire face à de nouvelles contraintes. Parmi celles-ci, une augmentation et une variabilité de la consommation énergétique avec une haute incertitude. Dans le cas présenté ci-dessus, nous pouvons expliquer la haute dispersion des mesures énergétiques, ainsi que le

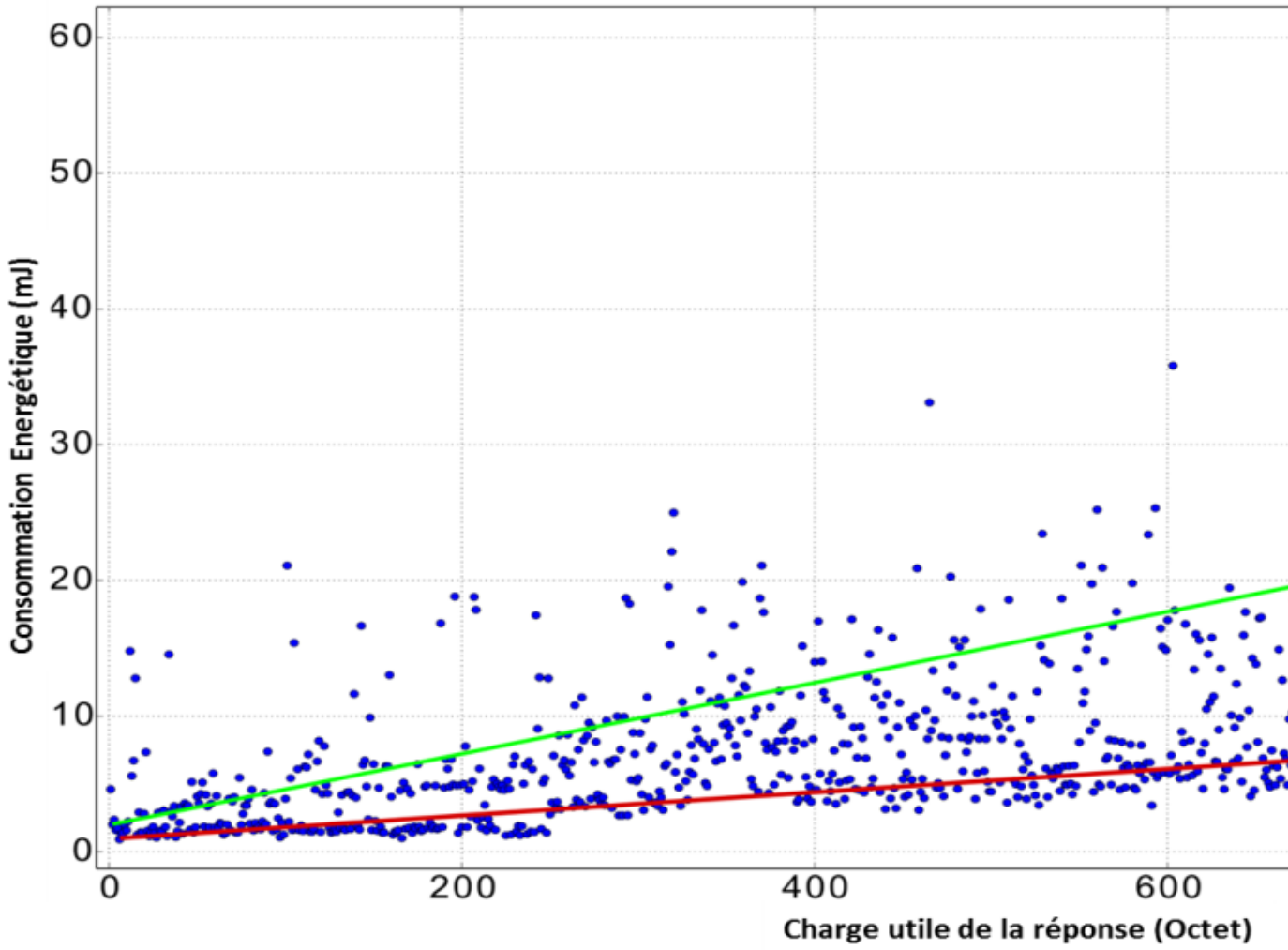


FIGURE 3.6 – Ecart-type de la consommation énergétique de l'objet du deuxième saut en environnement réel sur IPv4.

caractère imprédictible de l'écart-type associé, par le fait de l'existence de nombreuses interférences (radio, circuit) présentes en environnement réel.

En effet, ces expérimentations ont été effectuées dans nos locaux où nous pouvons dénoter la présence d'interférences WiFi et d'autres protocoles fonctionnant sur la même bande de fréquence que le 802.15.4 comme le Bluetooth. Ceci impacte le médium radio en causant des variations importantes dans la qualité du lien de communication radio. Cela, à son tour, augmente le taux de perte de paquets durant l'expérimentation, causant dans le cas de la pile uIPv4, une pénalité énergétique significative dû à la retransmission de bout en bout du paquet perdu (HTTP utilise le protocole de garantie de livraison TCP).

En plus des perturbations radios, nous pouvons aussi avancer l'existence d'éléments perturbateurs matérielles à l'intérieur de la plate-forme mesurée. Par exemple, une dérive d'horloge qui casse la synchronisation temporelle entre les objets, causant ainsi après un temps, une surconsommation énergétique significative. Tous ces types

d'interférences ajoutent un degrés de variabilité et d'incertitude dans l'estimation de la consommation énergétique d'un objet connecté. Elles sont donc de ce fait, difficilement modélisables pour être prises en compte par un simple simulateur d'objets connectés.

Par ailleurs, les "pics" de consommations énergétiques existants lors de la transmission de trames ayant une taille proche de la limite d'une trame 802.15.4 (i.e. chaque 48 octets de donnée utile), montre l'existence d'un réel "bug" énergétique préjudiciable à la durée de vie de l'objet. Un développeur qui voudrait naturellement et logiquement maximiser la contenance des trames envoyées provoquerait alors une surconsommation énergétique significative. Ce phénomène, visible sur les expérimentation en mode réel, identifie donc un problème matériel dans les Tmote-Sky ou dans le logiciel représenté par ContikiOS. Ce bug n'est pas du tout mis en avant par la simulation. Cela nous montre donc que la découverte et suivi de bugs ou incohérences énergétiques n'est pas toujours possible en milieu simulé.

Observations et analyses : IPv6

En ce qui concerne l'expérience en configuration IPv6 sous un environnement expérimental réel, Nous pouvons remarquer que le comportement des courbes de consommation énergétique des objets (illustré en figure 3.7) est tout aussi bruité que l'expérimentation précédente en IPv4. Les valeurs restent tout de même plutôt proches du cas simulé en IPv6, et peuvent être prédites et modélisées en utilisant un modèle presque similaire à celui utilisé dans le cas simulé. Néanmoins, nous observons le même phénomène que dans le cas IPv4 : une augmentation et une baisse subite et périodique de la consommation énergétique aux alentours de la taille maximale d'une trame.

L'écart-type, quant à lui est illustré en figure 3.8 comme exemple pour l'objet de deuxième saut. Il est caractérisé par plusieurs valeurs hautes approchant les 20 mJ ainsi qu'une dispersions des valeurs aux alentours des 10 mJ. Néanmoins, au global, l'écart type et donc la variation reste constante.

La cohérence et la ressemblance entre les résultats des cas simulés et réels en IPv6 peuvent être expliqués par l'efficacité énergétique de la pile uIPv6, et plus précisément la couche d'adaptation 6LoWPAN. En utilisant une caractéristique de la norme 802.15.4 permettant de multiples envois en rafales, cette couche permet la transmission de larges paquets IP directement entre nœud voisins. Cela réduit les effets néfastes en terme énergétique des interférences sur la perte de fragments radios.

La possibilité d'envoyer de larges paquets à travers le médium radio contraint (802.15.4) de voisin en voisin, réduit logiquement la probabilité de perte de paquets IP, et donc celle de retransmission de bout en bout du réseau. De plus, la possible perte d'un fragment radio possède un effet léger sur la consommation énergétique globale de la requête traitée par l'objet. La couche d'adaptation 6LoWPAN gèrera cette perte en effectuant une retransmission directe du fragment en un saut unique vers le voisin destinataire du paquet.

Malgré l'efficacité énergétique qui caractérise la pile IPv6 de Contiki, la problématique des interférences environnantes et la difficulté de modélisation de ces dernières se pose toujours. Nous remarquons d'ailleurs sur l'expérience IPv6 en environnement réel (Figure 3.7 et 3.8), une dispersion des valeurs d'écart type assez marquée. Aussi, plusieurs valeurs hautes de consommation énergétique sont présentes, bien que chaque expérience ait été rejouée 50 fois. Cela renvoie donc vers le même bug énergétique que celui observé et discuté dans le cas IPv4 en milieu réel. Ce phénomène est ici aussi totalement absent du cas simulé en IPv6. Nous observons donc

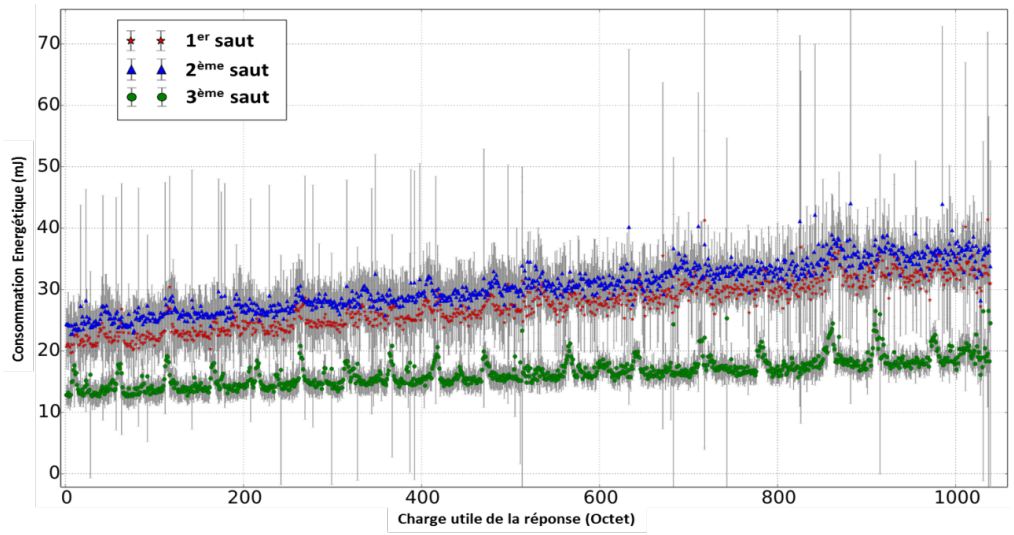


FIGURE 3.7 – Consommation énergétique des trois objets communicants du réseaux en environnement réel lors de l'utilisation d'IPv6.

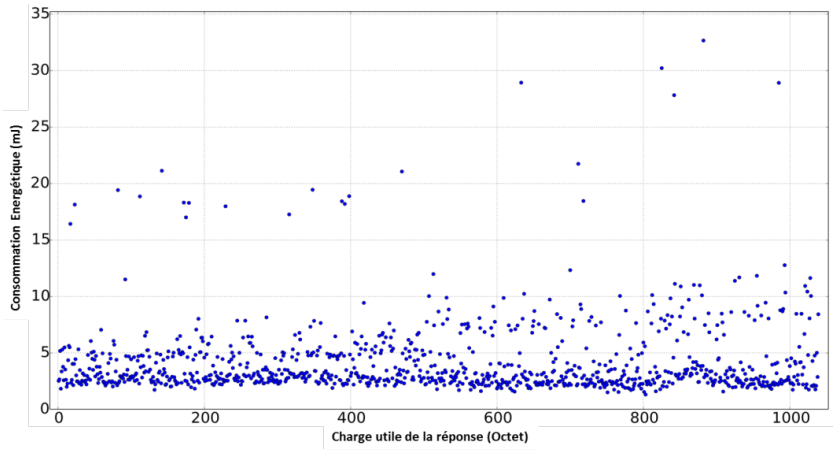


FIGURE 3.8 – Écart-type de la consommation énergétique de l'objet du deuxième saut en environnement réel lors de l'utilisation d'IPv6.

encore une fois les limites d'un environnement simulé pour l'obtention d'un profilage et suivi précis et fiable de la consommation énergétique d'un objet.

De plus, le fait d'utiliser la couche d'adaptation 6LoWPAN n'influe guère sur l'impact des interférences lors de la transmission de plusieurs paquets de données. Nous avons d'ailleurs fait l'expérience de bloquer le mécanisme de fragmentation de cette couche, induisant ainsi l'obligation d'envoi de multiple paquets IP lors de la demande d'une grande charge utile. Le résultat est comme attendu, quasi-identique avec les résultats de l'expérimentation en environnement réel en configuration IPv4, caractérisé ainsi par une haute consommation énergétique ainsi qu'une forte variabilité (écart-type).

3.5 Synthèse et conclusion

Ce chapitre nous a permis de présenter le profil et le comportement énergétique global d'une application web embarquée. Cette dernière est totalement adaptée à l'idée d'interopérabilité poussée par les paradigme de l'Internet des Objets et du Web des Objets. Nous avons expérimentalement évalué cette application ainsi que le réseau entier auquel elle appartenait selon deux configurations réseaux et deux environnements de déploiement. Grâce à cela, nous avons montré que dans bien des cas de la vie réelle, la simulation ne peut représenter un substitut de qualité à une expérimentation en environnement réel. Pour ne citer que quelques raisons, nous avons identifié les interférences radios présentes en environnement réel tel que le WiFi en tant que perturbations ayant un impact significatif sur la qualité du lien radio de communication 802.15.4 entre les objets. Cette détérioration du lien implique un accroissement non-négligeable mais surtout variable de la consommation énergétique de chaque objet du réseau. Spécifiquement, cela prend une ampleur plus importante lors de l'utilisation d'un protocole nécessitant une garantie de livraison des données car une retransmission des fragments perdus devient alors obligatoire. Par ailleurs, nous avons mis en exergues l'existence de bugs, ou tout du moins d'incohérences énergétiques non visibles lors des expérimentations en environnement simulé.

La simulation ne représente pas réellement l'entité à blâmer, car cette dernière repose sur un modèle qui doit être à même de retranscrire les impacts d'un environnement de déploiement réel. Néanmoins, ces impacts étant nombreux et variables dans le temps, la création d'un tel modèle de simulation qui serait aussi précis devient ardu et lourd à réaliser, ainsi qu'à répéter pour les nombreuses plates-formes embarquées hétérogènes existantes.

Afin de passer outre cette problématique, nous plaidons pour l'utilisation d'une méthode de profilage énergétique matérielle. Celle-ci étant capable de fournir des chiffres de consommation précis et réels tout en capturant les impacts de l'environnement de déploiement lors de l'exécution réelle de l'objet communicant. Nous pensons ainsi pouvoir aider les développeurs d'applications embarquées dans le cadre de l'Internet des Objets, à mieux concevoir leurs logiciels embarqués dans le but de connaitre et garantir une durée de vie maximale à l'objet connecté.

Troisième partie

Contributions

Chapitre 4

Réseaux cellulaires et consommation énergétique

Peu importe ce qu'on pourra vous dire,
les mots et les idées peuvent changer le monde. **Robin Williams, Le cercle des poètes disparus (1989)**

4.1 Introduction

Nous avons étudié lors du chapitre précédent, une technologie de communication courte portée basée sur la norme IEEE 802.15.4 qui régit aussi des technologies de communication tels que le Bluetooth. Cette norme s'applique très bien à des cas d'utilisation tel que les réseaux de capteurs ou encore des objets communicants utilisés dans le cadre de la domotique. Néanmoins, les contraintes de portée induites limitent fortement la généralisation vers d'autres scénarios d'utilisation nécessitant une architecture de communication distribuée et longue distance. Cette nécessité est souvent présente pour des besoins industriels concernant les projets M2M (*ang.* Machine to Machine).

Les technologies cellulaires offrent la possibilité d'effectuer des communications longue distance en profitant de la large couverture territoriale offerte par les infrastructures des opérateurs de télécommunication. Cela offre au système communicant des avantages indéniables en terme de connectivité réseau. A contrario, l'obtention d'une connectivité aussi large induit un prix important à payer, notamment en termes de consommation énergétique du module communicant.

4.2 Motivation

Il existe divers types et modèles d'objets communicants se différenciant selon divers critères. Nous pouvons citer les caractéristiques de performances que sont la mémoire embarquée, la puissance du microcontrôleur (microprocesseur) ou encore le facteur de forme de l'objet, qui va définir précisément les environnements de déploiement possibles. Le modem de communication embarqué afin d'établir la connectivité du système représente l'objet de notre étude dans ce chapitre. Nous nous intéressons donc aux objets communicants utilisant les technologies de communications cellulaires que sont la 2G, 3G.

Nos travaux étant totalement centrés sur la consommation énergétique des petits systèmes, les études et développements de ce chapitre sont axés sur l'efficacité et comportement énergétique des modules de communication cellulaire. Pour ces derniers, une gestion optimale de l'énergie consommée est primordiale afin d'assurer une durée de vie conséquente à l'objet et éviter des travaux de maintenance énergétiques causant en même temps un coût supplémentaire non négligeable pour un projet M2M.

Outre un niveau de consommation énergétique assez élevé, les modules de communication cellulaire sont aussi caractérisés par un comportement de fonctionnement plus complexe que la majorité des modems radios à l'instar des technologie IEEE 802.15.4. En effet, les technologies cellulaires ajoutent un nouvel aspect à la connectivité qui est l'opérateur du réseaux cellulaire. Ce dernier est en charge de la gestion des stations de base (*ang.* BTS : Base Transceiver Station) auxquelles les modems s'appairent, ainsi que du cœur de réseau de l'infrastructure. Suivant différentes politiques internes, l'opérateur peux influer, à tout moment, sur le modem cellulaire et donc en modifier certaines caractéristiques. Ceci induit alors un comportement variable du modem au fil du temps, complexifiant la gestion de l'efficacité énergétique de ce dernier.

La prise en main des modules de communication cellulaire est complexe. Néanmoins, malgré l'émergence de nouvelles solutions de communication pour l'IoT (e.g. LoRa, Sigfox, 4G-MTC), celles-ci ne peuvent se substituer aux réseaux cellulaires. En effet, sous une vue purement industrielle (qui nous concerne du fait du travail en collaboration avec la société Worlline), le recours aux technologies 3G ou même à la 2G - que beaucoup décrivent comme obsolète - reste un passage obligé. Le monde industriel évolue nettement moins vite que son pendant académique et recherche. Le remplacement des anciennes infrastructures ainsi que la mise en place de nouvelles représentent un coût significatif dont l'industrie ne peut/veut pas assumer. Ainsi, en dépit de la complexité de maîtrise des technologies cellulaires, leur utilisation intensive en industrie nous pousse à les étudier sous plusieurs angles, dont celui de la consommation et efficacité énergétique.

L'objectif de ce chapitre est d'apporter aux développeurs d'applications M2M un profilage de la consommation énergétique des communications cellulaires. Puis de leur fournir, basé sur ces profils, des guides de bonnes pratiques ciblés afin d'optimiser et d'arriver à un développement d'applications M2M efficace et économe en énergie.

4.3 Les réseaux cellulaires

Afin de pouvoir appréhender la consommation énergétique dans les communications cellulaires, il est nécessaire d'avoir une vue d'ensemble des infrastructures et des protocoles pilotant les communications cellulaires, d'autant plus que l'opérateur du réseau représente un acteur important dans le fonctionnement des modules de

communication 3G (UMTS/HSPA) et 2G (GSM/GPRS/EDGE). Nous présentons en Figure 4.1 l'architecture d'un réseau cellulaire UMTS. Celle-ci est par ailleurs similaire à l'architecture d'un réseau GPRS. Nous décrivons les principaux éléments du réseau :

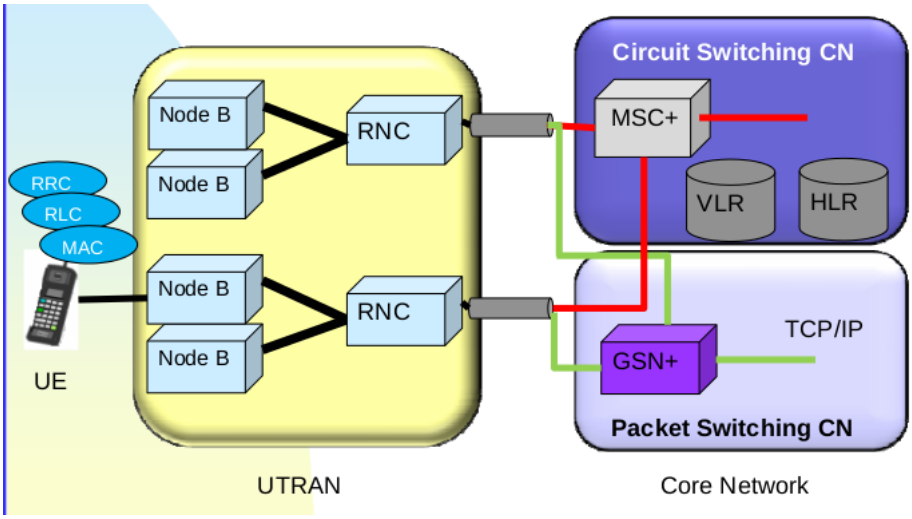


FIGURE 4.1 – Architecture d'un réseau cellulaire UMTS.

Équipement utilisateur (UE) il permet à l'application utilisateur de se connecter au réseau cellulaire. Il représente, le plus souvent, un téléphone ou dans notre cas, un objet communicant. Notre étude est centrée sur le modem cellulaire intégré à ces équipements ;

UTRAN réseau d'accès radio (*ang.* Universal Terrestrial Radio Access Network). Il représente la partie radio du réseau de télécommunication cellulaire. Il permet aux équipements utilisateurs d'accéder au réseau de télécommunication de l'opérateur. l'UTRAN regroupe deux composants distincts :

Node B la station de base (*ang.* BTS, Base Transceiver Station). Elle représente la balise radio de télécommunication. Cette balise appartient à l'opérateur de télécommunication et permet d'offrir une connectivité sur une superficie donnée. La superficie couverte par la balise radio est subdivisée sous forme de zones nommées "cellules". Chaque station peut gérer une ou plusieurs cellules. La station de base est identifiée comme l'élément le plus énergivore dans l'infrastructure cellulaire (UTRAN + CN) (voir Figure 4.2) ;

RNC *ang.* Radio Network Controller. Il contrôle les transmissions radio des stations de base. De ce fait, il contrôle toutes les ressources radio des Node B auxquels il est connecté. Le RNC gère la répartition de la ressource radio, le chiffrement des données avant l'envoi à l'équipement mobile. Le RNC assure les mécanismes de transfert intercellulaire (*ang.* handover) et de macro-diversité. Le transfert intercellulaire est la capacité du réseau à maintenir une communication lorsqu'un équipement utilisateur change de cellule. Le RNC prend une grande place dans notre étude. En raison de

son rôle de "gestionnaire de ressources", le RNC influe sur la consommation et le comportement énergétique du modem radio cellulaire connecté au Node B.

CN *ang.* Core Network. Il représente le cœur de réseau de l'architecture de télécommunication de l'opérateur. Elle est composée de deux grands domaines distincts : un domaine CS (*ang.* Circuit Switched) utilisé pour la téléphonie que nous n'aborderons pas ici. Ainsi qu'un domaine PS (*ang.* Packet Switched) qui permet la commutation de paquets (Mode DATA). Ce dernier est subdivisé en une passerelle permettant l'acheminement des données vers le cœur de réseau nommée SGSN (*ang.* Serving GPRS Support Node), ainsi qu'une passerelle aiguillant les paquets vers l'extérieur (i.e. Internet) nommée GGSN (*ang.* Gateway GPRS Support Node). Enfin, La structure de données permettant la session entre le SGSN et le GGSN est appelée contexte PDP (*Packet Data Protocol*). Le GGSN gère cette session PDP qui représente un contexte contenant les informations de QoS, login et mot de passe de l'utilisateur et assurant la communication sur l'Internet.

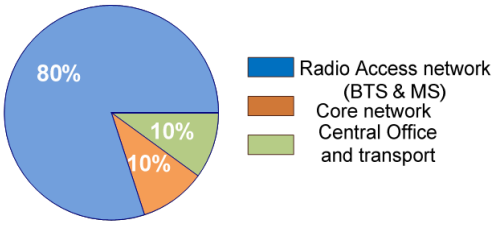


FIGURE 4.2 – Consommation énergétique d'une infrastructure d'un réseau cellulaire [Lou07].

4.4 Consommation énergétique des réseaux cellulaires

4.4.1 Comportement énergétique d'un modem de communication cellulaire en transfert de données

Dans la section précédente, nous avons défini l'architecture d'un réseau de communication cellulaire. Nous nous intéressons dans cette section au fonctionnement et au comportement énergétique des modules radio cellulaires de technologie 2G et 3G lors de communication entrainant un transfert de paquets de données.

Les modems de communication cellulaires fonctionnent suivant une machine à états bien définie (cf. Figure 4.3). Sur ce point, Les technologies 2G et 3G suivent d'ailleurs en théorie une machine à états semblable dans sa définition et dans les états qu'elle comporte [Eri99]. Cette machine à états pilote certes le comportement du module radio, néanmoins il est primordial de préciser que le composant régissant le fonctionnement de cette machine est le RNC. Ce dernier joue le rôle de coordinateur avec le module radio, en lui envoyant les commandes radio nécessaires afin que celui-ci puisse transiter à travers les différents états de la machine à états. Cette machine définit 4 états (dont 3 majeurs, illustrés en Figure 4.3), nous présentons chacun d'entre eux ci-dessous :

CELL_IDLE cet état représente le mode "veille" du module radio. Aucun échange radio n'est donc effectué dans cet état ;

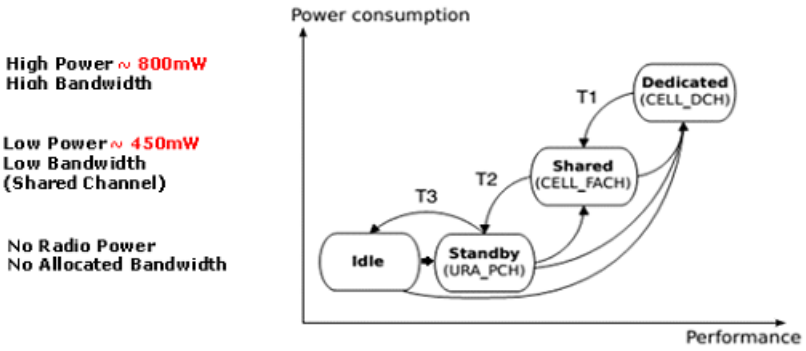


FIGURE 4.3 – Machine à états d'un module de communication 3G/UMTS [VNT12].

URA_PCH *ang.* Cell Paging Channel. Dans cet état, le canal radio est partagé par tous les équipements radio de la cellule de télécommunication. L'objet communicant (i.e. le modem cellulaire) surveille les messages de signalisation venant du contrôleur radio RNC via la balise radio (Node B) de la cellule. À l'instar du fonctionnement en état CELL_IDLE, la consommation énergétique du modem dans l'état URA_PCH est similaire à celle de l'état en CELL_IDLE. De plus, aucune activité radio ascendante (i.e. de l'objet vers le réseau de télécommunication) n'est possible dans cet état ;

CELL_FACH *ang.* Cell Forward Access Channel. Dans cet état, l'objet communique avec le réseau cellulaire via un canal spécifique partagé. L'objet peut envoyer une faible quantité de données à un débit assez bas. Le taux maximum des données que le modem peut envoyer dépend de l'engorgement du canal utilisé et est totalement défini par l'opérateur de télécommunication. De par son rôle de gestionnaire et de coordinateur, le RNC communiquera tout changement de ce seuil au modem de l'objet communicant utilisateur. Le niveau de consommation énergétique dans cet état est dit "moyen" (en théorie, 450 mW), certes bien plus grand que dans l'état "CELL_PCH" mais nettement moins par rapport "CELL_DCH" ;

CELL_DCH *ang.* Cell Dedicated Channel. Dans cet état, l'objet communicant se voit allouer un canal de transport de données totalement dédié dans les deux directions, ascendante et descendante. Cet état permet à l'objet d'atteindre un haut débit dans les transferts de données en ayant à sa disposition une grande bande passante. Néanmoins, cela est réalisé en échange d'une hausse significative de la consommation énergétique correspondant approximativement au double de la consommation énergétique dans l'état CELL_FACH (en théorie, 800 mW).

Dans la machine à états présentée précédemment, les transitions d'un état supérieur vers un état inférieur sont définies par le temps d'inactivité du module radio (T1, T2 et T3 sur la machine à états 3G représentée en Figure 4.3). Les valeurs T1, T2 et T3 représentent des timers d'inactivité. Le temps d'inactivité signifie ici, le temps pendant lequel aucun transfert de paquet radio n'est effectué. Ainsi, durant toute la durée de ce temps, le

module radio reste dans son état courant. Le module ne pourra rétrograder qu'à la fin de ce temps d'inactivité. Ce temps est plus communément appelé "temps de queue" (*ang.* Tail Time) [BBV09].

Il peut être considéré comme nuisible l'inclusion de ces timers d'inactivité dans la machine à états. Néanmoins, ce temps de queue possède une forte utilité, qui est notamment à l'avantage de l'opérateur de télécommunication. En effet, le temps de queue permet de laisser un laps temps au module radio cellulaire dans son état courant, afin que ce dernier puisse réagir rapidement dans le cas où un transfert de données se présenterait. Cela permet tout d'abord d'améliorer l'expérience utilisateur qui ne ressentira pas de latence significative à chaque transfert due à la latence du changement d'état. Cela vaut aussi tout autant pour un objet communicant autonome nécessitant un temps de réaction non négligeable en terme de communication. Avec une vue opérateur, le temps de queue permet d'atténuer le nombre de paquets de commandes qui sont obligatoirement échangés entre le module radio et le RNC lors d'un changement d'état. Ces échanges induisant une forte latence, leur réduction permet une forte atténuation et un ample allègement de la charge sur l'infrastructure réseau de l'opérateur de télécommunication.

Les transitions d'un état inférieur vers un état supérieur sont définies par le volume de données transférées. Ainsi, lorsque le module de communication radio cellulaire dépasse le quantité maximale de données pouvant être envoyées dans un état, le module cellulaire transite vers un état supérieur de la machine à états 3G. Afin de gérer et de mesurer en temps réel la quantité de données actuellement envoyées, un tampon est utilisé au niveau du module radio et du RNC. Ce tampon est nommé le RLC (*ang.* Radio Link Control) buffer.

Un point ne devant pas être négligé concerne la courte surconsommation énergétique présente lors des transitions entre état nommé énergie de rampe (*ang.* Ramp energy). Cette surconsommation d'énergie est dûe à l'activation de différents circuits matériels nécessaires au fonctionnement du modem dans le nouvel état.

Les transitions entre états s'effectuent via un protocole de communication nommé RRC (*ang.* Radio Resource Protocol) liant le modem cellulaire à l'infrastructure de l'opérateur [Eri99]. De ce fait, l'utilisateur du modem radio qui est dans notre cas, l'objet communicant via cette technologie radio, ne peut pas influer directement et explicitement sur l'état dans lequel se trouve le modem. Néanmoins, nous verrons dans ce chapitre, que l'objet communicant peut jouer sur la taille des paquets de données à envoyer et le timing d'envoi afin d'atteindre une bonne efficacité énergétique du modem de communication.

4.4.2 Inférence des paramètres RRC du réseau

Les deux paramètres RRC du réseau que sont, les timers d'inactivité ainsi que le tampon RRC représentent les principaux vecteurs influant sur la consommation énergétique du modem radio cellulaire :

— le temps de queue déterminant le temps pendant lequel le module de communication doit rester inactif afin de rétrograder d'un état et ainsi de réduire sa consommation énergétique ;

— le tampon de données RLC (*ang.* RLC Buffer) permettant de définir la taille maximale des données pouvant être envoyées avant de transiter vers un état énergétique supérieur.

En plus d'être entièrement définis et fixés par l'opérateur, ces deux paramètres sont variables au fil du temps et ne sont pas connus de l'utilisateur du modem. Ainsi, avant d'essayer d'effectuer des optimisations énergétiques des communication cellulaires, il est nécessaire de connaître ces paramètres. Pour cela, le plus

simple est d'utiliser un profilage énergétique qui nous permettrait d'observer facilement les états énergétiques du module de communication. Cette observation donne instantanément le comportement énergétique du module ainsi que les paramètres précédemment cités pilotant les transitions entre ces différents états.

Néanmoins, il n'est pas toujours possible de faire un profilage énergétique par cause de manque de moyens ou simplement de la difficulté de mise en place du profilage. De plus, les paramètres de transition peuvent être modifiés dynamiquement à tout moment par l'opérateur. Il est donc nécessaire d'avoir un moyen d'inférer ces paramètres dynamiquement après le déploiement effectif de l'objet.

À cet effet, une mesure qui permet d'inférer ces paramètres est le RTT (*ang.* Round Trip Time). Ce dernier décrit le temps d'aller et de retour entre deux points d'un réseau. Ainsi, le RTT permet de définir l'état actuel dans lequel se trouve le module cellulaire. Afin d'inférer les paramètres de transitions entre états via l'utilisation du RTT, nous pouvons nous baser sur les travaux de [VNT12] et utiliser les primitives suivantes :

— Temps de queue, transition CELL_DCH -> CELL_FACH : nous déclenchons une transition vers l'état CELL_DCH en envoyant un paquet de grande taille suivi par une séquence de petits paquets de taille beaucoup plus faible. À un certain temps, le module cellulaire rétrogradera vers l'état CELL_FACH. À ce moment-là, le RTT ayant la plus grande valeur indiquera qu'on est bien passé vers l'état CELL_FACH. Le temps de queue peut donc ainsi ainsi être mesuré ;

— Temps de queue, transition CELL_FACH -> CELL_IDLE : nous utilisons une approche de type dichotomique. Nous commençons par une estimation des bornes inférieure et supérieure du temps de queue. Nous choisissons une valeur de temps test incluse dans cette intervalle. Nous envoyons un paquet déclenchant une transition vers l'état CELL_FACH et nous attendons un temps équivalent au temps test choisi. Puis, nous testons le RTT de l'état où se trouve le module cellulaire afin d'ajuster les bornes précédemment définies. Si le module est dans l'état CELL_PCH, alors le temps de test choisi est plus grand que le véritable temps de queue. À l'opposé, si le module est dans l'état CELL_FACH, alors le temps de test est trop court. En itérant de cette manière le temps de queue de cette transition peut être calculé.

— Tampon RLC, transition CELL_PCH -> CELL_DCH : la méthode est similaire à celle utilisée pour le temps de queue concernant la transition de CELL_FACH vers CELL_IDLE. Nous choisissons ici deux bornes approximatives pour notre tampon RLC. Puis nous choisissons une taille de paquet de test pour le tampon RLC. Nous envoyons un paquet de cette taille-là, puis selon l'état dans lequel se trouve le modem cellulaire, les bornes précédemment choisies sont ajustées afin d'augmenter la précision de l'inférence de la valeur du tampon RLC ;

— Tampon RLC, transition CELL_FACH -> CELL_DCH : en ce qui concerne cette dernière inférence, nous commençons par envoyer une succession de paquets avec un large incrément afin de fixer rapidement un intervalle dans lequel une transition de CELL_FACH vers CELL_DCH a lieu. Nous attendons ensuite le temps de queue nous permettant de rétrograder vers l'état CELL_FACH, puis nous envoyons une succession de paquets avec de très petits incréments. Le but est d'ajuster et de trouver le plus précisément possible la taille de tampon permettant une transition de l'état CELL_FACH vers l'état CELL_DCH (cf.

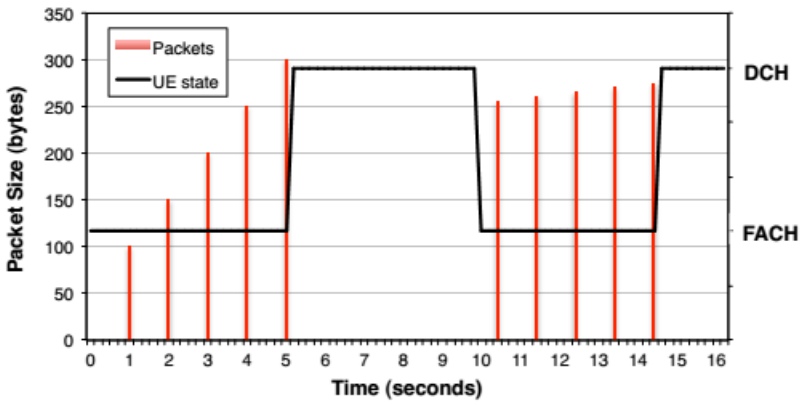


FIGURE 4.4 – Exemple d'inférence de la taille du tampon RLC pour une transition de l'état CELL_FACH vers CELL_DCH [VNT12].

Figure 4.4).

4.4.3 Optimisation énergétique dans les communications cellulaires de données

Parmi les deux paramètres précédemment discutés de la machine à états des modules cellulaires, le temps de queue est de loin le paramètre responsable de la plus grande part de surconsommation énergétique "inutile" lors des communications cellulaires de données (cf. Figure 4.5).

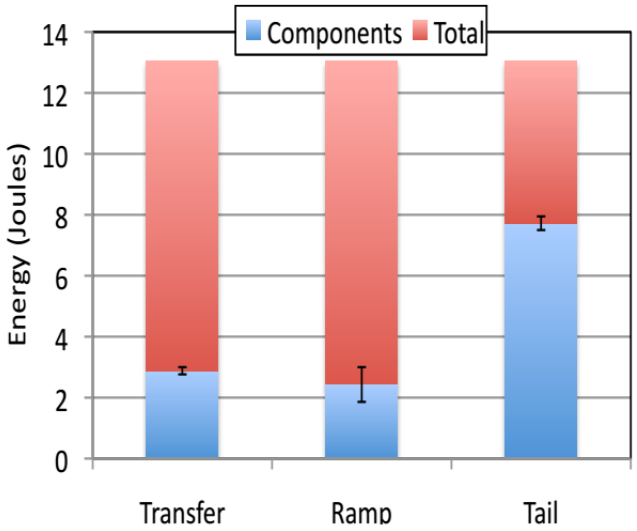


FIGURE 4.5 – Contribution du temps de queue à la consommation énergétique totale du module lors du transfert de 50 kilo-octets de données sur un module de communication 3G [BBV09].

Au vu de la forte contribution énergétique du temps de queue, il est primordial d'avoir des approches ayant pour but l'atténuation de cette surcharge énergétique. La première idée venant à l'esprit est d'agréger les paquets de données à envoyer ou à recevoir, le serveur pouvant lui aussi agréger les paquets devant être envoyés à l'objet. Comme nous pouvons le voir sur la Figure 4.6, plus les différents paquets à envoyer sont transmis sur un court intervalle, plus l'impact successifs du temps de queue est diminué, réduisant ainsi la consommation énergétique totale des envois. De ce fait, plus le temps séparant les transferts est long, plus l'énergie consommée est grande. Celle-ci est bornée par le temps de queue.

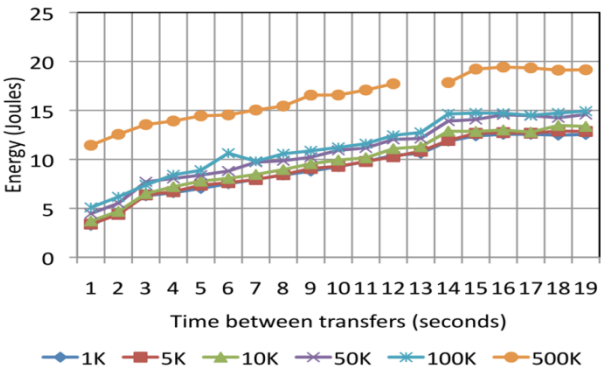


FIGURE 4.6 – Consommation énergétique de multiple envois de données en fonction du nombre de paquet envoyés, de leur taille et du temps entre chaque envoi [BBV09].

L'agrégation des données permet, dans beaucoup de cas, de réduire la consommation énergétique du module de communication cellulaire. Néanmoins, agréger des données signifie aussi retarder des transferts de données. Cela n'est pas toujours possible car des transferts prioritaires ou urgents ne pourront pas forcément être retardés. Par ailleurs, cela peut même devenir préjudiciable pour la consommation énergétique du transfert et ainsi avoir l'effet inverse à l'optimisation efficace voulue. Ce cas est décrit en Figure 4.7.

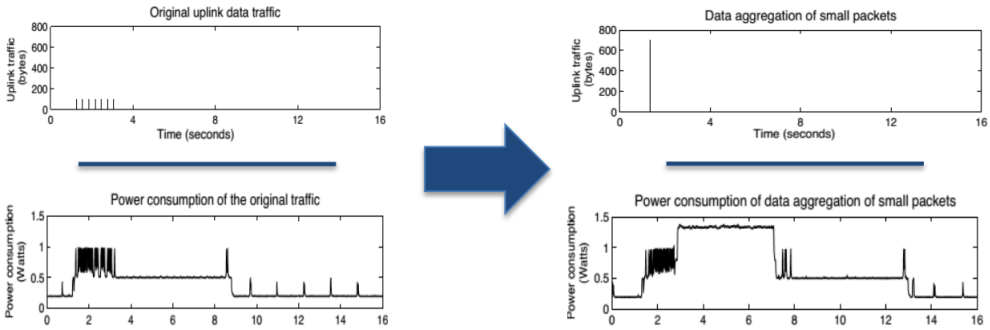


FIGURE 4.7 – Impact négatif de l'agrégation des données sur la consommation énergétique [VNT12].

Comme nous pouvons le voir, la consommation énergétique du transfert après agrégation est beaucoup plus haute que la consommation du transfert avant l'agrégation (envoi de plusieurs petits paquets séparément). Cela est dû au fait que l'envoi de l'agrégat des petits paquets déclenche une transition vers l'état CELL_DCH qui

représente l'état de consommation énergétique maximum. Cette transition n'est pas déclenchée si nous envoyons séparément tous les petits paquets avec un court laps de temps d'inter-transfert. Ce temps équivaut au mieux au temps nécessaire au RLC pour se vider.

Un algorithme prenant en compte cette contrainte de l'agrégation est présenté dans [VNT12]. Afin de toujours prendre la décision la plus économe en énergie, l'algorithme trie les transferts à effectuer par rapport au temps maximal pour lequel on peut les retarder. Quand un transfert doit être fait, on analyse, grâce à la connaissance (i.e. inférence) des paramètres RRC, si ce dernier est censé déclencher une transition vers l'état CELL_DCH. Si tel est le cas, nous en profitons pour envoyer tous les transferts stockés. Si le transfert déclenche une transition uniquement vers l'état CELL_FACH, alors un calcul et une analyse sont réalisés sur tous les transferts sauvegardés qui sont de taille assez petites afin ne pas déclencher une transition vers l'état CELL_DCH. L'analyse est simple et se base sur le choix du meilleur compromis. Est-ce plus économe en énergie de privilégier la basse consommation de l'état CELL_FACH malgré une bande passante très réduite ? Ou alors choisir la haute bande passante et vitesse de transfert de l'état CELL_DCH mais avec à contrario une forte consommation énergétique ? Si l'envoi de tous ces transferts est plus économe dans l'état CELL_FACH que dans l'état CELL_DCH, les transferts sont envoyés à la suite avec un court temps d'inter-transfert nécessaire pour vider le tampon RLC et ne pas déclencher de transition d'état. Sinon, tous les transferts sont envoyés d'un seul coup déclenchant ainsi une transition nécessaire vers l'état CELL_DCH.

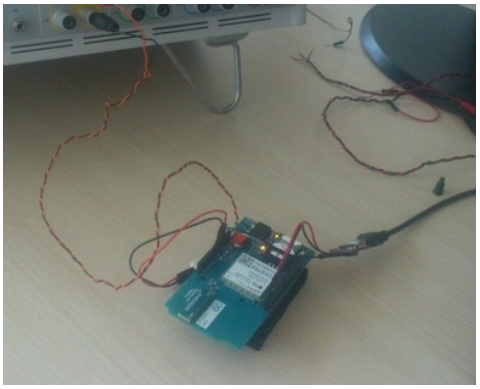
4.5 Retour d'expérience et guides du développeur

L'objectif central de la thèse est d'aider et assister le développeur à tenir compte de la contrainte énergétique lors du développement de l'application embarquée de l'objet connecté. Nous étudions donc maintenant le fonctionnement et comportement énergétique de plusieurs modules de communication 2G et 3G. Le but est de déduire et de fournir au développeur des "guides" de bonnes pratiques de développement lors de l'utilisation de modules de communication cellulaire.

Dans cette optique, Nous mesurons expérimentalement la consommation énergétique de quatre modules de communication (Figure 4.8) : Shield GPRS Arduino (Figure 4.8a), Modem GPRS Genpro18e (Figure 4.8b), MultiModem 3G rCell Router (Figure 4.8c) et une clé 3G Alcatel 500D (Figure 4.8d). Nous utilisons comme instrument de mesure électrique, l'analyseur de puissance Agilent N6705 que nous décrivons plus en détail dans le prochain chapitre.

Nous nous baserons et décrirons essentiellement dans cette section les expérimentations et conclusion tirées de l'utilisation de la clé 3G Alcatel 500D. Celle-ci est alimentée avec une tension fixe de 5 V. La clé 3G est un modem entièrement dédié à la communication cellulaire. Cela n'est pas toujours le cas des autres modems fournissant des fonctions de routage ou autres. Nous avons ainsi une plus grande confiance dans les valeurs énergétiques capturées.

Pour envoyer des informations en utilisant les moyens de transmission cellulaire, il existe deux types de méthodes bien distinctes. La première est l'utilisation des circuits GSM via les SMS (*ang.* Short Message System). Nous présenterons en premier cette méthode. La deuxième, que nous présenterons dans un second temps, se fait grâce à l'envoi d'informations encapsulées dans un paquet internet traditionnel. Néanmoins, nous



(a) Shield GPRS Arduino

(b) Modem GPRS Genpro18e

(c) MultiModem 3G rCell Router

(d) clé 3G Alcatel 500D

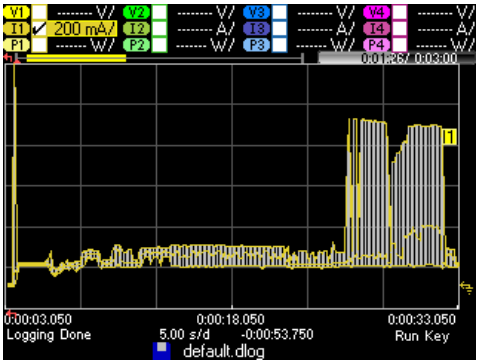
FIGURE 4.8 – Les modems cellulaires utilisés.

commençons par présenter la consommation énergétique des opérations de base nécessaires en amont de toutes communications

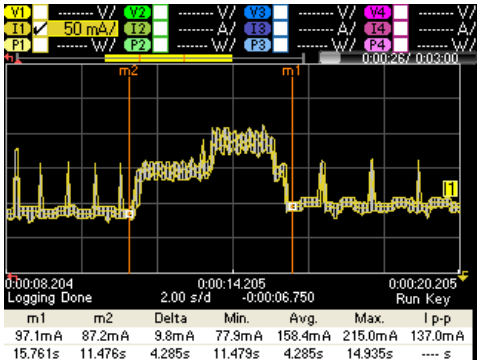
4.5.1 Consommation des opérations de base

Nous étudions ici la consommation énergétique des opérations de base. Les opérations de base désignent les primitives élémentaires et nécessaires afin que le modem cellulaire puisse initier et effectuer des communications cellulaires de données ou GSM. Ces primitives concernent aussi l'état du module de communication : allumé, éteint, appairé à un réseau, etc.

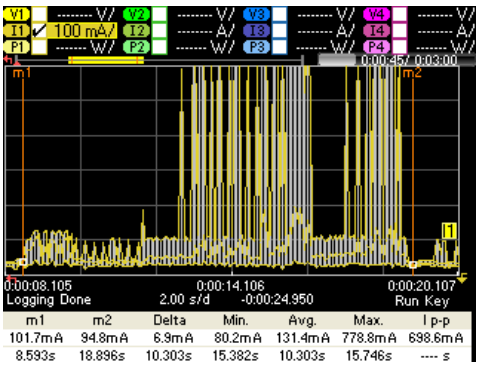
Une des principales tâches est de mesurer la consommation énergétique des primitives qui sont importantes lors d'un scénario M2M. On conviendra que le module de communication, à l'instar de l'objet lui-même, ne fonctionne que périodiquement avec des périodes de transmission et des périodes de veille/arrêt. Les primitives dont nous avons mesuré la consommation énergétique sont partiellement illustrées en Figure 4.9. Les graphes de consommation présentés ici et dans le reste du chapitre sont directement tirés de l'analyseur de puissance Agilent N6705 qui fait office ici d'instrument de mesure. Ces graphiques présentent l'intensité du courant consommée par



(a) Allumage du modem.



(b) Switch de GPRS vers UMTS.



(c) Switch d'UMTS vers GPRS.

FIGURE 4.9 – Quelques échantillons de mesures des opérations de base.

le module en fonction du temps. L'énergie est calculée facilement dans un second temps en prenant en compte le Voltage qui est fixe. L'échelle des graphiques est directement fournie par le terme seconde/division présent.

Nous donnons ci-dessous ainsi que dans le tableau 4.1, un listing complet et précis du profilage énergétique des primitives de base testées.

— Allumage du modem (cf. Figure 4.9a) ;
— Appairage et désappairage aux/des réseaux GSM, GPRS, UMTS (2G et 3G) ;
— Établissement et suppression d'un contexte PDP et d'une connexion ppp (Protocole Point à Point) ;
— Basculement entre deux réseaux de différentes technologies (GPRS <-> UMTS) (cf. Figure 4.9b et Figure 4.9c).

Après expérimentation, les consommations énergétiques moyennes ci-dessous ont été calculées concernant les primitives de base du modem cellulaire :

— **Allumage du modem** : 17.5 Joules (durée : 25 secondes). Il est à noter que l'allumage propre du modem ne concerne que le temps nécessaire pour la stabilisation et la mise en fonctionnement des circuits composant le modem. Il n'y pas d'appairage effectif associé à un réseau au moment de l'allumage ;
— **En fonctionnement normal/IDLE** (lorsqu'il n'y a aucune opération apparente à effectuer) : le module consomme 480 milliwatts en instantané. Sur une durée témoin de 10 s de fonctionnement, cela résulte en une consommation de 4.8 J ;

Opération	Allumage	fonctionnement	Appairage			Désappairage	*PDP context*	Connexion ppp	Suppression PDP Context	switch	
			GSM	GPRS	UMTS					UMTS → GPRS	GPRS <-> UMTS
Consommation (en joule)	17,5	4,8	0,5	6,5	3,5	<1	8,6	8,6	8,6	6,5	3
Temps (en seconde)	25	10	<1	10	4		9	9	9	10	3,7

TABLE 4.1 – Tableau récapitulatif des expérimentations sur la consommation des opérations de base.

- **Appairage aux réseaux** :
  - **GSM** : 0.5 J (durée : < 1 s) ;
  - **GPRS** : 6.5 J (durée : 10 s), nécessite un appairage au réseau GSM ;
  - **UMTS** : 3.5 J (durée : 4 s), nécessite un appairage au réseau GSM ;
- **Désappairage d'un réseau** : un échange de signalisation qui est faible en consommation énergétique et pouvant donc être négligé ;
- **Établissement et Suppression d'un contexte PDP** : 8.6 J (durée : 9 s) ;
- **Basculement entre deux réseaux GPRS <-> UMTS** :
  - **UMTS <-> GPRS** : 6.5 J (durée : 10 s) ;
  - **GPRS <-> UMTS** : 3 J (durée : 3.7 s).

L'une des premières constatations concerne la similarité dans le fonctionnement d'un côté énergétique après et avant une primitive de base. Ainsi, par exemple, la consommation énergétique reste équivalente avant et après l'appairage aux réseaux GPRS ou UMTS. Il en est de même avec quasiment toutes les autres primitives de base décrites précédemment. Néanmoins nous observons une légère différence lorsqu'il s'agit de l'appairage au réseau GSM. Malgré tout, cette différence d'une valeur 8mA est trop faible en comparaison avec la consommation en courant moyenne d'un module cellulaire (=94mA) , pour donner lieu à un véritable choix à faire lors de l'appairage ou non à un réseau GSM.

Guide 1 : Il n'y pas de calcul ou de choix à réaliser lors d'une connexion à un réseau ou lors de l'établissement d'un contexte de données (i.e. PDP). Ainsi, mise à part la consommation de la primitive en elle-même, il n'y a pas de surconsommation liée au changement d'état du module de communication (appairé au réseau GPRS, appairé au réseau UMTS, etc.).

Guide 2 : Nous avons conclu précédemment qu'il n'y avait pas de calcul ou de choix crucial à faire lors d'une connexion à un réseau ou lors de l'établissement d'un contexte de données. Néanmoins des décisions importantes peuvent être réalisées concernant le fait d'éteindre ou non le module de communication. En effet, l'arrêt du module de communication à un temps "t" induira automatiquement lors du prochain envoi de données à "$t+1$", l'allumage de ce dernier puis le ré-appairage à tous les réseaux nécessaires : GSM pour l'envoi ou la réception de SMS et GSM + GPRS/UMTS pour les transfert de données.

Avec un simple calcul, nous listons les trois initialisations possibles du modem cellulaire lors d'un besoin de communication en mode SMS ou en mode de donnée :

— Allumage + GSM = 18 J (durée : 26 s)
— Allumage + GSM + GPRS + contexte PDP = 33.1 J (durée : 45 s)
— Allumage + GSM + UMTS + contexte PDP = 30.1 J (durée : 39 s)

La consommation instantanée en fonctionnement IDLE est dans notre cas de 0.48 Watts. De cela, nous pouvons écrire l'équation de l'énergie consommée E (Joules) en fonction du temps (t) : $E(t) = 0.48 * t$.

Afin de décider s'il est profitable d'éteindre le module de communication, il suffit de savoir dans combien de temps s'effectuera le prochain transfert de données. En prenant comme exemple, une communication de données sur un réseau UMTS (3G), nous obtenons l'inégalité décrite en équation 4.1.

$$E(t) = 0.48 * t_{IDLE} < 30.1J \tag{4.1}$$

Exemple direct : Pour envoyer des données sur un réseau UMTS, il nous faudra approximativement 30 J pour mettre en place tous les appairages nécessaires, comme indiqué précédemment. Dans ce cas précis, si ce transfert est à effectuer dans un délai inférieur à 62 s (cf. inégalité 4.1), alors il est préférable en terme de consommation énergétique et de latence du transfert, de ne pas éteindre le module de communication.

4.5.2 Consommation énergétique des SMS

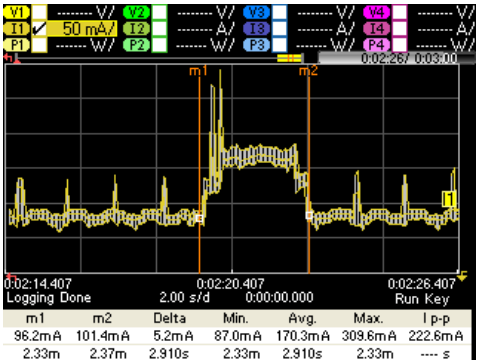
Le SMS est un moyen fondamental en communication cellulaire pour envoyer de la donnée. Le SMS utilise dans la quasi-totalité des cas, la signalisation GSM (donc le réseau GSM) pour faire transiter les données. Dans la majorité des scénarios, le SMS est utilisé pour faire transiter de très petites quantités de données, ce qui peut être très intéressant ainsi qu'important pour les applications M2M nécessitant une faible bande passante. En plus de cette utilisation, le système de SMS peut aussi être utilisé afin de réveiller le modem cellulaire d'une veille ou encore lui envoyer des commandes systèmes pour mettre à jour son fonctionnement interne.

Nous avons effectué plusieurs envois et réceptions de SMS de différentes taille en utilisant le modem/clé 3G Alcatel 500D. La taille du texte pouvant être contenu dans un SMS varie de 1 caractère à 160 caractère. Nous utilisons bien entendu un autre interlocuteur afin d'envoyer et recevoir ces SMS. Celui-ci est un simple téléphone portable, l'objectif est de faire transiter les SMS en provenance/en direction du modem Alcatel par le réseau GSM. Les résultats médians de ces expérimentations sont illustrés en Figure 4.10. Nous résultats sont les suivants :

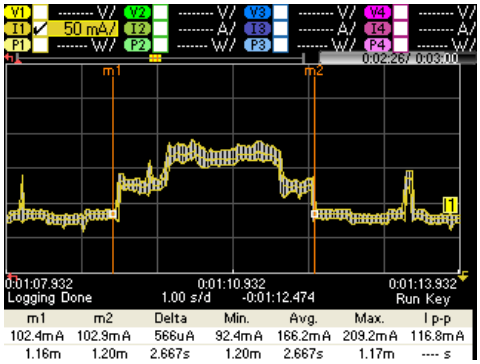
— nous obtenons une consommation moyenne de 2.4 Joules pour l'émission d'un SMS de 160 caractères (taille maximale) et 2.1 Joules pour la réception d'un SMS de même taille (cf. Figure 4.10) ;
— La consommation énergétique, par rapport au volume de données transférées, est fixe pour chaque octet de plus, tant que ce volume est inférieur à la taille maximale d'un seul SMS (160 caractères).

Des observations précédentes nous tirons ces conclusions :

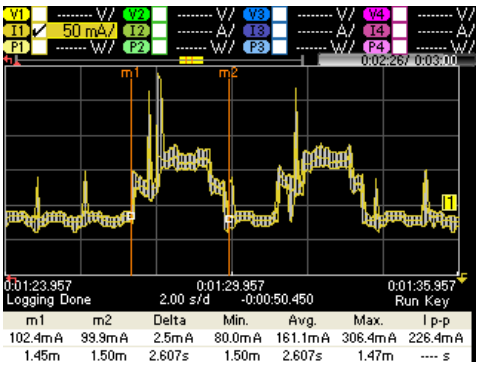
— La consommation énergétique en émission et en réception de SMS diffère tout de même de 0.3 J, indiquant ainsi une nette variation énergétique entre flux montant et flux descendant ;
— La consommation énergétique d'un transfert de SMS est linéairement liée aux nombres de SMS transférés et non pas au nombre de caractère (octet) contenu dans celui-ci (cf. 4.10).



(a) Envoi d'un SMS de 160 caractères.



(b) Réception d'un SMS de 160 caractères.



(c) Réception d'un SMS de 250 caractères. Nous remarquons parfaitement les deux consommations énergétiques distinctes des deux SMS composant le SMS de 250 caractères.

FIGURE 4.10 – Consommation énergétique pour l'émission et réception d'un ou plusieurs SMS.

Un facteur important pouvant drastiquement changer la consommation énergétique des transferts de données via SMS concerne la compression du texte du SMS [FRP+06]. En utilisant une compression adaptée, il est possible de réduire le nombre de SMS à envoyer, baissant ainsi grandement la consommation énergétique global du transfert. La compression utilisée devra évidement être une compression sans perte, n'utilisant aucun dictionnaire et optimisée pour des textes composés de peu de caractères à l'instar des SMS. Enfin, il est à signaler que dans tous les cas, l'énergie consommée pour effectuer la compression/décompression est nettement inférieure à l'énergie économisée en réduisant le nombre de SMS et d'octets envoyés. Autrement-dit, la consommation du CPU de l'objet connecté est d'un ordre de grandeur inférieur à celui de la transmission de par le modem cellulaire. L'utilisation de la compression est donc toujours adéquate.

Guide 3 : La consommation énergétique des SMS est linéairement liée au nombre de SMS transférés et non pas au nombre d'octets contenus dans le SMS. De ce fait, le premier but est de retarder les opérations de transfert de SMS afin de pouvoir agréger les données afin de remplir au maximum l'espace libre d'un SMS. En deuxième étape, la compression peut être prévue afin de réduire le nombre de SMS envoyés.

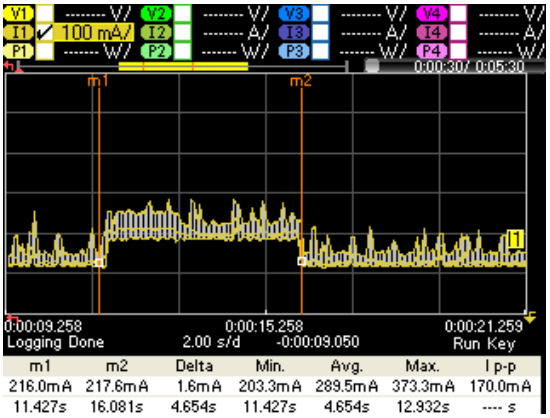
4.5.3 Consommation des communications en données

Nous avons présenté dans la première partie de ce chapitre, différentes optimisations pouvant être appliquées sur des communications cellulaires en données (Mode DATA) pour en améliorer l'efficacité énergétique. Ces optimisations sont toutes étroitement liées à la machine à états du protocole RRC régissant le comportement des modems de communication cellulaire tant sur un plan performance que énergétique.

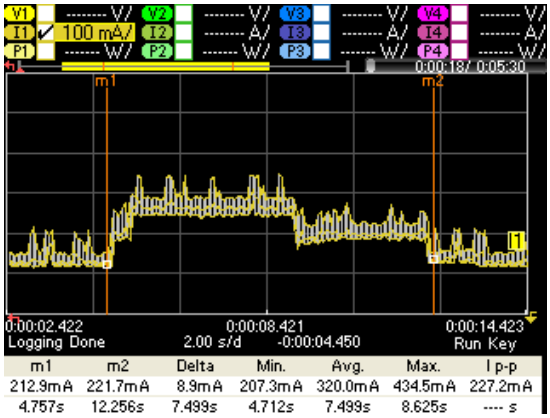
L'optimisation la plus indécise est certainement celle concernant le choix à faire concernant l'agrégation ou non des transfert de données. On pourrait aussi en parler de manière dual en choisissant ou non de décomposer un transfert de données à effectuer en plusieurs sous-transferts de taille moins grande. Chaque transfert est séparé par un court laps de temps équivalent au temps nécessaire pour que le cache RLC se vide. Ce cache est la mesure principale utilisée par le contrôleur RNC pour faire transiter ou non le modem vers un état de haute performance et de haute consommation énergétique. Par exemple, Dans notre cas, ce temps est égal à 0.6 s. Ce paramètre est fixé par l'opérateur du réseau et non pas par l'utilisateur, l'application ou le modem cellulaire.

Nous illustrons dans la Figure 4.11, un exemple de cas où l'agrégation des transferts possède un impact néfaste sur l'efficience énergétique du modem cellulaire. Les sous-figures 4.11a et 4.11b décrivent respectivement la consommation en courant du MultiModem 3G sur un transfert fragmenté et un transfert agrégé de 800 octets de données. Comme nous pouvons facilement nous en apercevoir, le fait d'avoir agrégé les 2 transferts à envoyer a déclenché une transition vers l'état CELL_DCH multipliant la consommation énergétique d'un facteur deux par rapport à l'envoi fragmenté des données.

Néanmoins, l'idée d'agréger tous les transferts en un seul et unique envoi n'est pas une idée fausse. En effet, une partie significatives des cas et scénarios peuvent bénéficier de cette agrégation en termes de consommation énergétique. La Figure 4.12 illustre un de ces cas. Ainsi, l'envoi ici d'un paquet de grande taille (i.e. 5000 Octets), en un seul transfert, résulte en une meilleure efficacité énergétique que lors de sa décomposition en plusieurs sous transfert. La prise en compte de cette contrainte reste toute de même importante. L'algorithme décrit plus tôt permet une réduction de la consommation énergétique du module cellulaire d'environ 50 %. Ce type d'algorithme peut ainsi être d'un impact positif dans le cas du M2M où les tailles des transferts de données sont souvent avoisinantes des limites des tampons de transition d'état RLC.

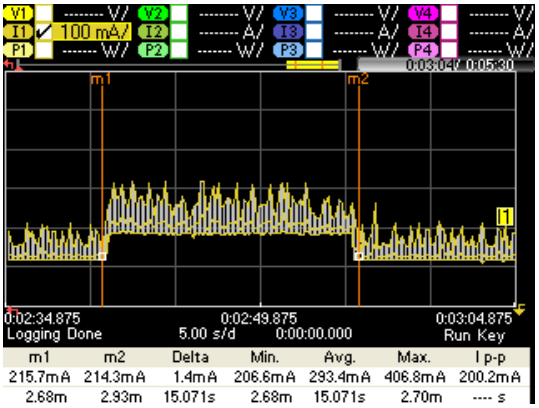


(a) Envoi de 2 paquets de données de 400 Octets chacun. Energie consommée équivalente à 12 Joules.

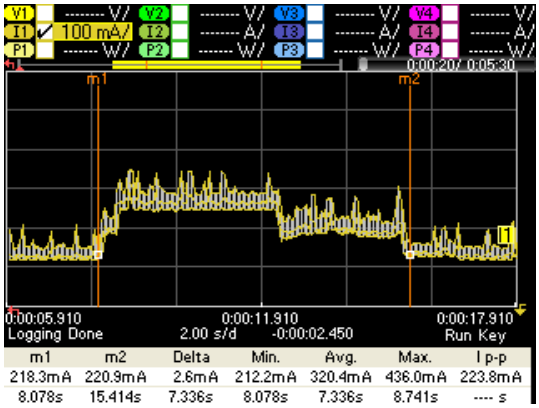


(b) Envoi d'un unique paquet de données de 800 Octets. Energie consommée équivalente à 21 Joules.

FIGURE 4.11 – Mesures montrant un cas où l'agrégation de transfert est préjudiciable pour la consommation énergétique



(a) Envoi de 10 paquets de données de 500 Octets chacun. Energie consommée équivalente à 39.5 Joules.



(b) Envoi d'un unique paquet de données de 5000 Octets. Energie consommée équivalente à 23 Joules.

FIGURE 4.12 – Bénéfices de l'agrégation des données pour la consommation énergétique

Guide 4 : "Est-ce que je dois agréger ou plutôt subdiviser les transfert de données à effectuer ?" : Au final, la décision d'agréger ou non les données à envoyer via le modem cellulaire est une décision à prendre en connaissance de tous les paramètres influant sur la consommation énergétique d'un module cellulaire. Ces paramètres décrit précédemment dans ce chapitre sont tous les paramètres appartenant au protocole RRC et intervenant dans les transitions d'état au sein de la machine à états du modem cellulaire. Cette connaissance permet de définir approximativement le nombre maximum de paquets pouvant être envoyés séparément avant que l'agrégation devienne l'optimisation la plus efficace en énergie.
De cette observation, nous déduisons aussi qu'il peut être plus que bénéfique de fragmenter soi-même un gros transfert de données et revenir donc à l'un des cas présentés précédemment. Néanmoins, pour que cela soit possible, il est nécessaire que les données applicatives du transfert soit facilement fragmentables où alors qu'on introduise un mécanisme simple de fragmentation dans le protocole applicatif permettant ainsi au serveur de pouvoir reformer le paquet de données d'origine effectué par l'objet communicant.

4.5.4 Impact de la latence des transferts sur la consommation énergétique

Lors d'un transfert de type requête/réponse entre l'objet connecté et une entité distante (e.g. serveur, un autre objet connecté.), la latence de réponse de l'entité distante induit un impact négatif sur la consommation énergétique. En effet, comme nous l'avons vu précédemment, la consommation énergétique d'un module cellulaire est largement induite par les timers d'inactivité contrôlant le passage d'un état vers un autre de consommation énergétique moindre ou plus élevé.

Lors de la survenu d'une latence dans une communication, l'impact énergétique négatif du temps de queue sera plus élevé selon la longueur de cette latence. Cependant, il reste complexe de formuler mathématiquement l'impact énergétique de la survenu d'une latence dans un transfert de données. En effet, il peut y avoir une multitude de cas et de scénario impliquant cette latence. l'hypothèse la plus commune est le temps mis par un serveur pour répondre à une requête venant du modem. En simplifiant et réduisant ces cas, nous pouvons quantifier l'énergie perdue à cause d'une latence de la façon suivante :

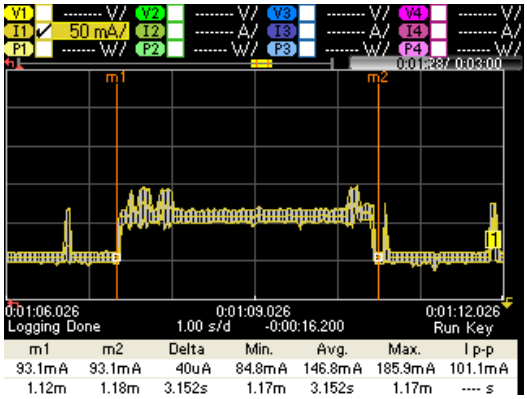
— Si on est dans l'état CELL_DCH lors de la survenu de la latence :
Énergie consommée = min(Latence, Temps de queue en état CELL_DCH) x Puissance énergétique en état CELL_DCH
Dans notre cas d'étude, cela équivaut à : E = min(Latence(s), 3.9 s) x 1.125 W = 4.3 Joules (Si Latence > temps de queue de CELL_DCH) ;

— Si on est dans l'état CELL_FACH lors de la survenu de la latence :
E = min(Latence(s), Temps de queue en état CELL_FACH) x Puissance énergétique en état CELL_FACH
Ce qui équivaut dans notre cas d'expérimentation à : E = min(Latence(s), 3.2 s) x 0.73 W = 2.3 Joules (Si Latence < Temps de queue en état CELL_FACH) ;

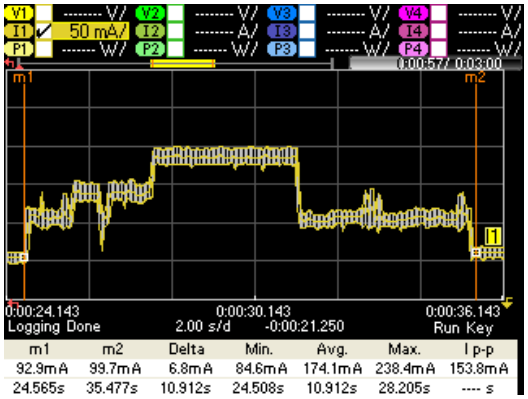
— Si on est dans l'état CELL_DCH et que nous avons totalement épuisé les timers d'inactivités. Après la fin de la latence, le transfert induit, déclenche une transition vers l'état CELL_DCH :
E = min (Latence(s), Temps de queue en état CELL_DCH) x Puissance énergétique en état CELL_DCH
+ min (Latence(s), Temps de queue en état CELL_FACH) x Puissance énergétique en état CELL_FACH
L'application de cette équation à notre cas résulte en une énergie consommée de 6.8 Joules.

La latence induite peut donc au maximum être égale à 6.8 Joules (sans prise en compte de l'énergie de rampe). Cette valeur représente évidement une importante surconsommation préjudiciable en terme d'efficacité énergétique pour l'objet connecté.

Un exemple pratique de cette surconsommation est donné ci-dessous en 4.13a. On peut observer l'impact de la latence sur une simple connexion TCP. La Figure 4.13a présente une connexion TCP fiable avec une latence négligeable. La Figure 4.13b décrit une connexion TCP où une latence a causé la retransmission successive de l'acquittement de la poignée de main TCP induisant ainsi, un temps de connexion plus long mais aussi une transition non prévue vers l'état de consommation maximal CELL_DCH. De ces événements, se traduit une consommation énergétique plus élevée d'un facteur quatre par rapport à une connexion TCP sans latence (i.e. une latence négligeable).



(a) Connexion TCP saine avec une latence négligeable. Énergie consommée équivalente à 2.3 Joules.



(b) Connexion TCP avec une latence d'une seconde. Énergie consommée équivalente à 9.5 Joules.

FIGURE 4.13 – Impact d'un temps de latence de réponse sur la consommation énergétique d'une simple connexion TCP.

Guide 5 : Comment remédier à ce problème de latence ? Malheureusement, il n'y a pas vraiment de solution toute faite à ce problème. Dans la plus part des cas, le correspondant d'un objet connecté utilisant un module cellulaire sera un serveur distant d'une infrastructure IT (i.e. Infrastructure traditionnelle composée d'une grappe de serveurs).
Réduire la latence équivaudra à optimiser au mieux la communication entre le serveur et l'objet connecté, et cela particulièrement sur le serveur. En effet ce dernier devra traiter en priorité les requêtes venant des objets afin d'y répondre le plus rapidement possible et empêcher ainsi une surconsommation énergétique induite par la latence de communication/réponse.

4.5.5 Flux descendant et flux montant

Dans les descriptions de fonctionnement du protocole cellulaire 2G/3G, il est souvent noté le fait que la bande passante allouée pour le flux descendant est plus grande que celle allouée pour le flux ascendant. Inversement, la consommation énergétique de la transmission de la même quantité de données est plus grande dans le cas d'un flux ascendant que d'un flux descendant.

Au fil de nombreuses expérimentations sur cette question, il n'a été observé aucune différence franche et notable entre la consommation énergétique en flux descendant et en flux ascendant, il en est d'ailleurs de même pour le temps de transfert. Néanmoins, il existe entre le flux ascendant et le flux descendant une différence notable dans la taille limite du tampon RLC utilisé pour transiter vers un état de consommation énergétique supérieur.

Dans le cadre de nos expérimentation, nous trouvons ainsi que le tampon RLC pour une transition état de CELL_PCH vers CELL_DCH est :

— En Flux descendant : 900 Octets ;
— En Flux Ascendant : 501 Octets.

Ainsi, nous arriverons plus souvent et rapidement dans l'état de consommation maximum CELL_DCH lors de l'utilisation de flux ascendants que dans le cas de flux descendants. Il est aussi nécessaire de spécifier que la SIM utilisée pour nos expérimentations est une SIM M2M. De ce fait, cette SIM ainsi que le fonctionnement qui lui est réservée par l'infrastructure réseau de l'opérateur de télécommunication peut avoir été spécifiquement conçue pour équilibrer la bande passante du flux ascendant et flux descendant.

4.6 Synthèse et conclusion

Dans ce chapitre, nous avons présenté une étude réaliste de la consommation énergétique des communications cellulaires 2G et 3G. À travers diverses expérimentations nous avons établi des guides de bonnes pratiques qui peuvent venir en aide aux développeurs concevant des applications M2M contraintes en énergie. Les technologies cellulaires, de par les infrastructures déjà existantes, sont faciles à utiliser et à incorporer dans un nouveau cas d'utilisation. Ce type de communication est donc souvent privilégiés des industriels pour des raisons de temps de mise sur le marché. Avec le positionnement industriel de notre travail de thèse, nous pensons que notre étude ainsi que les guides fournis peuvent venir en support du développeur embarqué lors de la conception de son application connectée afin d'atteindre une meilleur efficacité énergétique.

L'étude décrite dans ce chapitre peut aussi être vu comme une introduction nécessaire par rapport aux prochaines innovations dans le domaine des technologies cellulaires pour le M2M. En effet, lors de l'année 2017, nous avons vu l'émergence de la technologie 4G-LTE-MTC (*ang.* LTE for Machine Type Communication). Ce standard vise à incorporer les exigences du M2M et de l'IoT directement dans la définition de la norme et de la technologie 4G-LTE. Parmi les changements apportés, nous pouvons citer : la possibilité d'utiliser des bandes de communication étroite afin de limiter la consommation énergétique de l'objet communicant ; ou encore l'aptitude laissé au développeur de directement agir sur le fonctionnement du protocole RRC (ce qui n'est pas possible en 2G, 3G et 4G).

À travers la nouvelle technologie de communication 4G-LTE-MTC ou encore la 5G qui arrivera d'ici quelques années, un fonctionnement de plus en plus complexe est à appréhender par le développeur. Autrement dit, les machines à états qui sont au cœur du fonctionnement de chaque module de communication cellulaire deviennent de plus en plus enrichies par de nouvelles fonctionnalités. Le développeur se doit de maitriser ces possibilités afin d'optimiser l'efficacité énergétique de son application.

La question qui se pose est de savoir si nous pouvons venir en support du développeur de manière automatisée, afin de l'aider à appréhender la consommation énergétique de son application ? Ou plus encore, expliquer au développeur le comportement énergétique interne de son application embarquée pour lui permettre de voir les points d'amélioration et la meilleur façon d'atteindre une efficacité énergétique maximale.

Chapitre 5

Proposition : un environnement de profilage énergétique d'applications embarquées

A notre époque de communications à la vitesse de l'éclair, de télémachins
et autres trucs finissant en "el", une lettre, c'est devenu rare
Jean Dion

Dans ce chapitre, nous décrivons la première étape de notre solution. Celle-ci vise à venir en aide et en support au développeur d'applications embarquées lors de la conception d'applications contraintes en énergie pour des petits objets connectés. Partant des constatations de nos expérimentations du chapitre précédent, nous proposons au développeur une méthode de profilage énergétique du logiciel embarqué basée sur une véritable mesure de la consommation physique. Nous détaillons dans ce chapitre, les différentes composante de notre proposition de profilage énergétique.

5.1 Introduction

Nous l'avons répété à plusieurs reprise durant ce document : l'Internet des Objets est en pleine expansion. De plus en plus d'objets connectés font leur apparition et s'intègre aujourd'hui dans divers facettes de la vie quotidienne (e.g. domotique, structure civile ou voiture). Bien-sûr, cela vient avec une augmentation des fonctionnalités intégrées ou plutôt embarquées sur ces objets physique. Ces derniers ne sont au final, que de simples systèmes embarqués auxquels on adjoint différents composants périphériques afin d'assurer une interaction et une communication avec l'environnement.

Afin d'assurer les fonctionnalités de ces objets, de l'intelligence doit donc être embarquée sous forme d'une application logicielle. A ce point, les développeurs doivent déjà gérer les contraintes de calcul et de mémoire de ces petits systèmes afin de pouvoir embarquer toute l'application responsable des fonctionnalités de l'objet. Cependant, dans le cas des objets ne disposant pas d'une alimentation continue et fonctionnant donc grâce à des batteries, la tâche devient plus ardue. En effet, la consommation énergétique se rajoute comme nouvelle contrainte avec laquelle le développeur doit jongler. Contrairement aux autres, cette contrainte ne tolère que très peu d'erreur de la part du développeur. La mauvaise estimation de la consommation énergétique peut provoquer l'arrêt du fonctionnement de l'objet après sa mise en route. Cela peut être contrariant voir irréversible lorsque le changement de la batterie est difficile voir même impossible (e.g. un objet enfoui dans le béton pour la surveillance de structures [SA07]). En outre, une mauvaise appréhension de la consommation énergétique de son application peut aussi avoir des effets énergétiques néfastes à une échelle globale comme le démontre l'agence internationale de l'énergie (IAE) [Age14].

De par ces constats, afin de gérer au mieux la contrainte énergétique, nous pensons fortement que le développeur doit avoir une vue globale et précise de la consommation énergétique de son application. Nous proposons donc de l'aider dans cette tâche via la contribution décrite dans ce chapitre. Notre contribution ici, est la proposition d'un cycle développement centré sur la contrainte énergétique. À travers ce cycle, nous construisons un framework de profilage énergétique du code source embarqué d'une application logicielle. Celui-ci produit des résultats de profilage à un grain fonctionnel ainsi qu'une visualisation graphique de ceux là à l'intention du développeur. Il est basé sur une instrumentation du code source de l'application et la mesure de consommation de celle-ci via une technique matérielle de mesure.

5.2 Motivation et fondement : une approche basée sur le matériel et le réel

Le cycle de développement décrit dans ce chapitre possède comme principal objectif d'aider le développeur d'applications embarquées. Par ailleurs, les bases et fondement de cette proposition découlent des observations faites et décrites dans le chapitre *"état de l'art"* et le chapitre *"simulation Vs monde réel"*. Le point le plus important dans notre réflexion est l'importance de profiler la consommation énergétique du système dans des conditions réelles de fonctionnement et non pas via une simulation basée sur un modèle approximatif qui ne correspond pas vraiment aux attributs matériels du système. L'utilité de ce mode de profilage réside dans la captation des éventuels impacts pouvant être induits par le monde réel (environnement externe du système). Ces impacts sont difficiles, voir impossibles à résumer et à unifier à travers différents environnements.

Le deuxième pilier de notre réflexion est une extension du précédent. En effet, afin d'assurer l'obtention de chiffres énergétiques aussi précis que possible, il est nécessaire de ne pas se limiter à une approche de mesure logicielle (e.g. Powertrace), mais d'utiliser une technique de mesure matérielle basée sur une plate-forme de mesure électrique. Cette approche permet de directement fournir de manière physique la grandeur correspondante à la consommation énergétique du système, et ainsi être fidèle à la dissipation énergétique effective ayant lieu.

Néanmoins, l'inconvénient pouvant subsister avec cette approche de profilage concerne le développeur. En effet, celui-ci se retrouve à devoir gérer l'utilisation d'une plate-forme matérielle de mesure en plus de la

corrélation de la consommation énergétique avec le code source de son application embarquée. Les développeurs de systèmes embarqués n'ayant pas toujours les aptitudes et connaissances afin d'appréhender la consommation énergétique de leur système, nous proposons de les aider et leur faciliter cette tâche à travers notre cycle de développement et le framework de profilage associé.

La principal motivation et objectif réside dans l'automatisation et la transparence lors de l'utilisation du framework grâce à une instrumentation automatique du code source de l'application embarquée. Par ailleurs, l'ajout d'un retour visuel permet au développeur une compréhension simple des chiffres énergétiques reçus. Le tout permet une simplicité de rejeu du profilage énergétique après modification interne de l'application.

5.3 Un cycle de développement centré sur la consommation énergétique

Dans cette contribution, nous proposons dans un premier temps un environnement de développement centré sur la contrainte énergétique. En effet, nous pensons et affirmons qu'en plus d'un profilage énergétique précis, il est aussi nécessaire de centrer le développement de l'application embarquée sur la problématique de la consommation énergétique.

Nous présentons et illustrons notre cycle de développement en Figure 5.1. La phase critique de la mesure énergétique est placée au cœur du cycle, qui repose en entier dessus.

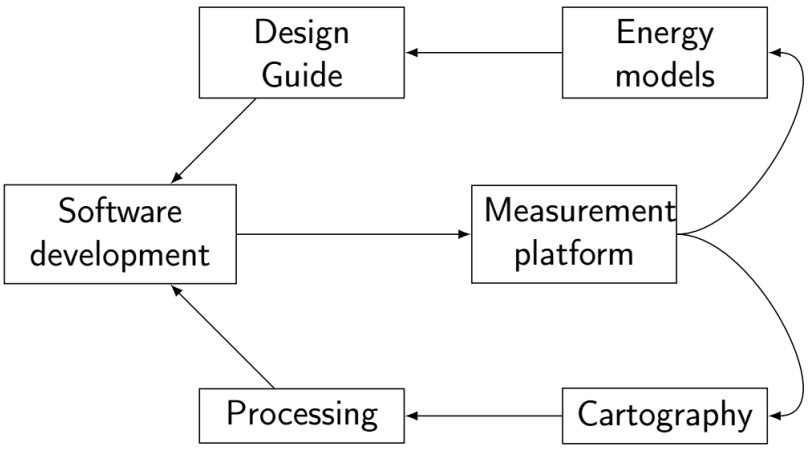


FIGURE 5.1 – Le cycle de développement centré sur la ressource énergétique.

Nous pouvons décomposer l'architecture de notre cycle de développement en deux parties. La première est la partie inférieur. Dans celle-ci, une cartographie énergétique du code source (i.e. l'application logicielle du développeur) est effectuée. Un retour visuel de la cartographie au niveau fonctionnel est présentée au développeur. Dans ce retour, chaque fonction du code source de l'application est exposée avec sa part de contribution à la consommation énergétique globale. À la suite de cette boucle, le développeur dispose des données de consommation énergétique lui permettant de modifier ou non son application en conséquence. En cas de modification, un rejeu est alors réalisé, un nouveau cycle de profilage est lancé afin d'observer les conséquences énergétiques.

Les processus rentrant en jeu lors de cette phase du cycle de développement sont illustrés dans en Figure 5.2. Plus précisément, L'ensemble de ces processus représente l'implémentation du cycle de développement dans ce que nous appelons "*le framework de cartographie énergétique*". Ces processus sont expliqués plus en détails dans la suite de cette section.

Afin de résumer et donner une vue d'ensemble du framework, nous pouvons directement donner un exemple d'utilisation. Ainsi, pour un système et application embarquée donnée, le code source interne de celle-ci passe par une étape d'instrumentation automatique. Le binaire provenant de la compilation du code source instrumenté est transféré sur la plate-forme embarquée cible. Puis, durant l'exécution de l'application sur le système, celui-ci est électriquement mesuré via une plate-forme de mesure énergétique matérielle. Une fois cette mesure terminée, les trace énergétiques sont exploitées afin de fournir les détails de consommation de chaque fonction instrumentée dans le code source de l'application sont affichés au développeur. Celui-ci peut alors modifier sont application et effectuer un autre tour de profilage énergétique de son application.

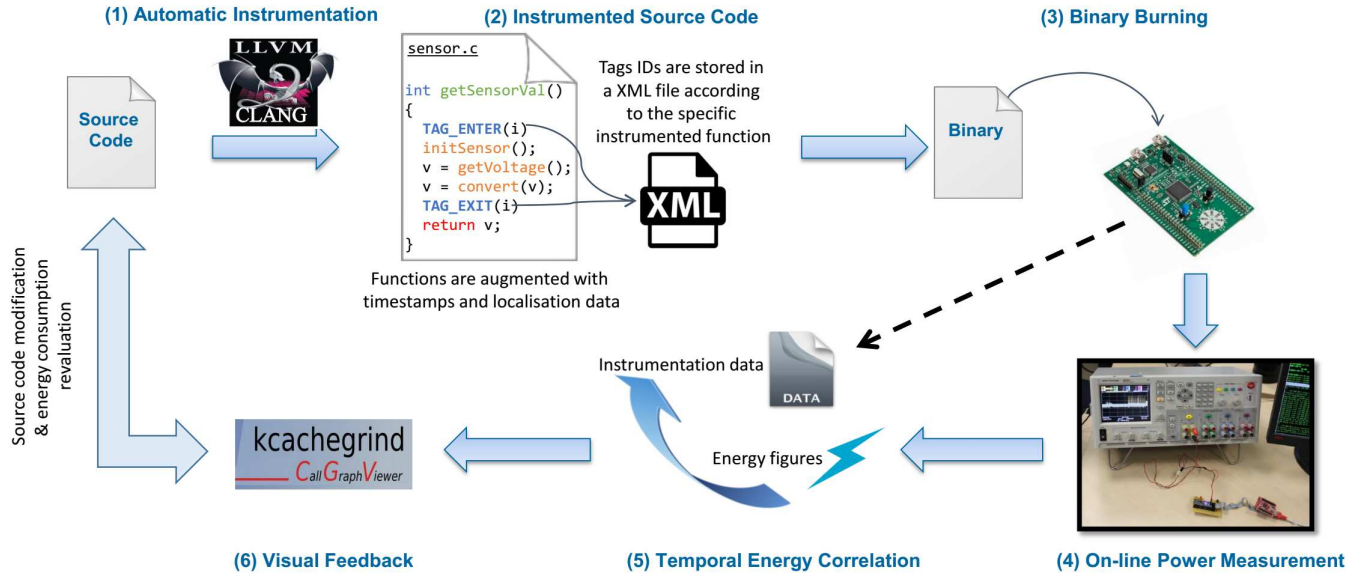


FIGURE 5.2 – Le schéma suivi pour effectuer le profilage énergétique d'un code source embarqué.

La deuxième et dernière partie du cycle de développement présenté est la partie supérieure de la Figure 5.1. Celle-ci se réfère au processus d'inférence de modèles énergétiques présentés dans le chapitre 6. Les modèles énergétique générés reflètent et résume le comportement énergétique d'un composant périphérique du système embarqué. La création de modèles énergétiques est une tâche difficile et longue. La phase de création elle-même nécessite une longue période d'acquisition d'échantillons énergétiques. Afin d'atténuer la difficulté de cette tâche, nous reposons le processus de génération de modèles énergétiques sur le mécanisme de profilage et de cartographie présenté ci-dessus. En mettant à profit le framework de cartographie énergétique, il est possible de faciliter la réalisation de diverses passes de mesure énergétique et donc la collecte de plusieurs échantillons de consommation du système embarqué sous différentes conditions. Les échantillons collectés sont ensuite traités

à l'aide d'outils statistiques afin d'extraire les informations pertinentes sur le comportement énergétique du système embarqué.

En ayant connaissance que les opérations d'entrées/sorties sont les composantes les plus consommatrices en énergie dans un petit système embarqué, nous utilisons ce cycle de génération de modèles sur les composant périphériques embarqués. Les modèles spécifiés résument le comportement énergétique des périphériques cibles, notamment les états énergétique et les transitions entre états mise en évidence par les échantillons énergétiques acquis. Enfin, la conclusion de notre cycle de développement se réfère à la génération de guides pour un développement efficace en énergie. Ainsi, en collectant et en traitant les modèles énergétique produits, nous pouvons théoriquement concevoir des guides spécifiques de développement énergétique concernant la plate-forme matérielle du système embarqué et l'application exécutée dessus. Ces guides peuvent être textuels ou alors même être impacté sur le code source du développeur embarqué. Les travaux possibles concernant la génération de ces guides de développement représentent les travaux futures de cette thèse.

5.4 Instrumentation du code source embarqué

La première étape de notre framework est l'instrumentation du code source de l'application embarquée. Cette étape représente un bloc critique et primordial de notre cycle de développement et le framework associé. En effet, se reposer uniquement sur les traces énergétiques fournies par la mesure matérielle n'est pas suffisant afin de déduire la consommation énergétique de chaque fonction logicielle. Ainsi, nous introduisons cette instrumentation de code afin d'obtenir des informations logiques de la part de l'application embarquée en cours d'exécution et en cours de mesure.

L'objectif principal de nos travaux est de faciliter au développeur la phase de profilage énergétique. Dans ce but, nous concevons la procédure d'instrumentation du framework afin qu'elle puisse être automatique et transparente du point de vue du développeur. Nous utilisons la bibliothèque de compilation offerte par le compilateur CLANG afin de concevoir une application tierce responsable de l'instrumentation du code source embarquée de l'application (voir *2.1* en Figure 5.2).

Clang [co] est une interface de compilation pour le compilateur LLVM. Clang peut effectuer des analyses syntaxiques sur tout code source basé sur un langage C (C, C++, ObjectiveC). En outre, Clang offre plusieurs API (LibClang, LibTooling, Clang Plugin) simples d'utilisation afin de guider cette analyse syntaxique et la modification du code source si nécessaire. En résumé, Clang analyse le code source et forme AST (Arbre syntaxique abstrait, ang. *Abstract Syntax Tree*) qui - comme son nom l'indique - est une représentation abstraite du code source. Néanmoins, il reste beaucoup plus simple de travailler avec cette représentation pour effectuer l'analyse. Le choix de Clang se justifie par le fait que l'API fournie permet de relier facilement chaque structure de donnée de l'arbre abstrait (e.g. *FunctionDecl*, *ReturnStmt*) à la représentation réelle dans le code source associé.

Nous utilisons Clang afin d'effectuer une analyse syntaxique de code source de l'application embarquée afin d'instrumenter les fonctions ciblées. Pour chaque fonction instrumentée, deux tags y sont insérés. Un premier tag au début de la fonction et un deuxième tags à la fin (i.e. avant chaque déclaration de retour ou avant l'accolade fermante). Les fonctions à instrumenter peuvent être spécifiées par le développeur si celui-ci cible précisément certaines fonctions. Le framework peut aussi réaliser une instrumentation plus étendue. Ainsi, il est possible de

rechercher de manière récursive et d'instrumenter toutes les sous-fonctions appelées par celles spécifiées par le développeur. Ainsi il est possible d'observer la consommation énergétique d'une seule branche d'appel parmi toutes celles possibles. Tout le code source n'est donc pas inutilement instrumenté si cela n'est pas désiré par le développeur. Il est à noter que Clang est utilisé uniquement pour l'analyse syntaxique et l'instrumentation du code source original de l'application embarquée. Le code produit après instrumentation est lui compilé via le compilateur GNU-GCC associé au type de microcontrôleur sur la plate-forme. Celui-ci est transféré après sur la plate-forme cible pour l'expérimentation (voir *2.3* en Figure 5.2).

Lorsqu'une fonction est instrumentée, deux tags sont donc insérés en entrée et en sortie de celle-ci (voir *2.2* en Figure 5.2). Chaque tag effectue une estampille temporelle et est responsable de stocker l'adresse mémoire de la fonction actuelle ainsi que l'adresse de la fonction appelante. Les données logiques résultantes de ces tags d'instrumentation insérés sont enregistrées directement dans un tampon de la mémoire centrale du système embarquée cible (i.e SRAM). Une fois le tampon rempli, celui-ci est vidé et les données contenues sont envoyées via l'interface série du système cible à la vitesse la plus haute possible. Nous avons choisi d'utiliser cette interface pour effectuer le transfert car elle représente l'interface la plus disponible sur la majorité des systèmes embarqués. L'utilisation d'une sonde JTAG aurait pu être aussi possible afin d'atteindre des vitesses de transfert bien plus grande. Néanmoins, la disponibilité de broche d'interfaçage ainsi que de module de débogage JTAG n'est pas assurée sur toute les plates-formes embarquées déjà assemblées et livrées aux développeurs. De plus, cela oblige à l'utilisation d'une sonde JTAG haute performance induisant une problématique de coût d'achat significatif ainsi qu'une limitation dû à l'aspect propriétaires de ces sondes (e.g. IAR, Segger, etc.).

Parmi les données générées à partir des tags, les estampilles temporelles ont une grande importance car elle sont utilisées pour localiser sur les traces énergétiques de la mesure, l'entée et la sortie de chaque fonction instrumentée (voir *2.5* en Figure 5.2). Afin de générer ces estampilles, il est nécessaire d'utiliser un compteur matériel. Plus la résolution du compteur est précise, plus la corrélation avec la trace énergétique sera correcte. Pour exemple, nous utilisons sur les plates-formes embarquées de type ARM (à base de microcontrôleur Cortex-M) le compteur matériel lié directement à l'horloge cadençant le processeur (*DWT Cycle Counter*). Ce compteur est ainsi caractérisé par une résolution au cycle près. Sur des plates-formes embarquées avec un micro-contrôleur ne possédant pas un compteur aussi précis, un Timer peut être utilisé en contrepartie d'une perte de résolution de la corrélation énergétique entre fonctions logicielles et consommation énergétique. La mesure énergétique via l'instrument matériel ainsi que le compteur matériel utilisé sur la système embarqué cible sont tous deux synchronisés, au début de l'expérience, par un GPIO numérique.

Chaque tags inséré via la procédure d'instrumentation implique un code source supplémentaire. Celui-ci induit donc un surcoût temporelle et énergétique. Cependant, dans notre cas de figure, les tags insérés conduisent à une faible perturbation du comportement original de l'application embarquée. Cela est dû au peu de données générées à travers chaque tag, ainsi que le peu d'instructions exécutées pour y arriver. Ainsi, chaque exécution d'un tag consomme 20 cycles (Plus ou moins 3/4 cycles à cause du pipeline ARM Cortex-M). Le véritable problème peut concerner le transfert du tampon mémoire à travers l'interface série. Sur ce point, il est à noter que l'interface série est une interface consommant très peu de courant (2mA, mesuré de manière empirique sur les cartes embarquées testées). Enfin, connaissant la durée du transfert, ainsi que le moment et l'endroit où il a eu lieu, la sur-consommation énergétique est simplement retranché de la consommation énergétique mesurée au moment du transfert.

5.5 Plate-forme matérielle de mesure énergétique

Dans le cycle de développement que nous proposons, l'étape de mesure de la consommation énergétique représente un composant principal. De ce fait, le framework que nous décrivons dans ce chapitre est soutenu par une approche de mesure matérielle basée sur une plate-forme de mesure de consommation énergétique (voir *2.4* en Figure 5.2). Nous présentons dans cette section, deux plates-formes matérielles de mesure de la consommation énergétique. La première, est un instrument industriel de laboratoire, l'analyseur de puissance électrique "*Agilent N6705A*". Celui-ci est l'instrument qui a été utilisé durant la quasi-totalité de notre travail de thèse. La deuxième plate-forme est une création conjointe matérielle et logicielle de notre part. Celle-ci est en cours de finalisation, mais les différentes caractéristiques, dont l'évolutivité et la flexibilité, en font un moyen de mesure prometteur pour la suite de travaux connexes.

5.5.1 Analyseur de puissance Agilent N6705A

L'Agilent N6705A est la plate-forme de mesure énergétique que nous avons utilisé tout au long des travaux de thèse décrits dans ce document.

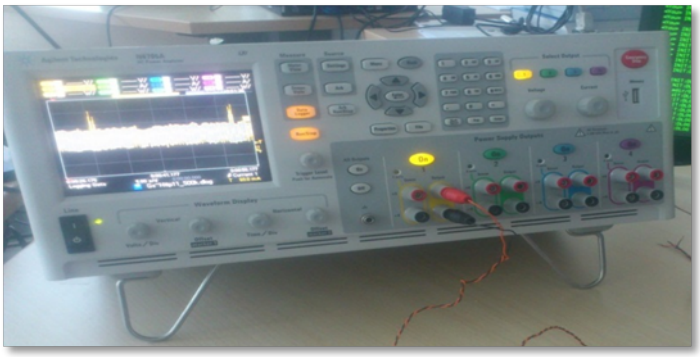


FIGURE 5.3 – Analyseur de puissance Agilent N6705A

L'Agilent N6705A illustré en Figure 5.3 est à proprement dit, un analyseur de puissance électrique (ang. *Power Analyzer*). Nous utilisons cet analyseur dans nos travaux comme plate-forme de mesure de l'intensité du courant consommée par le système sous étude. En effet, comme nous l'avions expliqué lors du chapitre état de l'art, dans la portée de nos travaux, l'intensité du courant (I) est suffisante pour déterminer la consommation énergétique d'un objet quelconque. Cela étant dû par le fait que la majorité des objets connectés fonctionnent sous une tension fixe (c.f. $Energy = CourantxTensionxTemps$).

Ainsi, si on limite l'utilisation de l'instrument aux capacités de mesure du courant, nous pouvons citer trois de ses caractéristiques qui sont critiques pour nos travaux :

— Une fréquence d'échantillonnage maximale de 50KHz lors de l'utilisation d'une seul voie de mesure ;
— Une précision de mesure pouvant aller jusqu'à 0.03 % + 15 μ ;
— Une résolution de mesure pouvant atteindre 18 bits.

L'Agilent N6705A prend en charge une large gamme de configurations de tension et de courant qui sont suffisantes pour couvrir la plupart des exigences d'alimentation des dispositifs embarqués. Il est d'ailleurs

possible de sélectionner - si connu à l'avance - un intervalle englobant la dynamique du courant, permettant ainsi d'améliorer la précision de la mesure résultante. L'Agilent expose 4 sorties numériques permettant de mesurer jusqu'à 4 charges d'alimentation distinctes. Cela permet de profiler en profondeur la consommation d'énergie d'un système embarqué en mesurant séparément ses périphériques externes.

Concernant la prise en main et l'utilisation, l'instrument peut être interfacé et commandé à travers une interface réseau Ethernet. Plus précisément, l'établissement d'une simple connexion TCP permet l'envoi de différentes commandes prédéfinies permettant entre autre de lancer une mesure en fixant divers paramètres (e.g. la durée, la dynamique si connu, les voies à échantillonner). La récupération des échantillons de mesure, une fois celle-ci terminée, s'effectue aisément via une méthode similaire de communication et d'interfaçage avec l'analyseur de puissance. Ces échantillons sont présentés sous forme de fichiers de type csv (ang. *Comma-Separated Values*), contenant les valeurs d'intensité du courant pour chaque voie physique échantillonnée.

Ces fichiers représentent les traces énergétiques traitées dans l'ensemble des outils, durant ce travail de thèse. Il nous semble donc primordial de donner une définition formelle d'une trace énergétique :

Definition. (Trace énergétique) Les traces énergétiques résultantes d'une phase de mesure peuvent être vues comme des séries temporelles (ang. *time series*).

Une série temporelle résultante d'une trace énergétique est une série finie de couples (t_i, c_i) representant les échantillons de la mesure. t_i représente le temps (en Seconde) ou l'estampille temporelle de l'échantillon de mesure. c_i décrit l'intensité du courant mesuré (en Ampere) de l'échantillon. Les attributs d'une série temporelle énergétique sont :

- $i \in \mathbb{N}, \forall i \in [0, L]$, où L représente la longueur de la série temporelle : $t_i \in \mathbb{R}$ and $t_i < t_{i+1}$
- $i \in \mathbb{N}, \forall i \in [0, L] : t_{i+1} - ti = T$ où T représente la période d'échantillonnage de la mesure énergétique (c.f. La fréquence d'échantillonnage est directement décrite par f tel que : $f = \frac{1}{T}$)
- La durée de la phase de mesure d (en Seconde) est directement obtenu par : $d = L.T$

La consommation énergétique E de toute la trace énergétique produite ou alors d'une sous-partie de celle-ci peut être facilement calculée. Pour une sous-partie décrite par l'intervalle $[t_a, t_b]$, nous calculons la consommation énergétique E (en Joules) par : $E_{t_{ab}} = \int_{t_a}^{t_b} P(t)dt$, avec $P(t) = c(t).V(t)$ Dans la grande majorité des cas étudiés sur les objets connectés, la tension (en Volt) est toujours fixe. De ce fait, la trace énergétique suffit pour le calcul de la consommation énergétique. La précédente formule se retrouve donc simplifié : $E_{t_{ab}} = V \int_{t_a}^{t_b} c(t)dt$.

Malgré les avantages indéniables de l'outil qui représentent un apport conséquent dans nos travaux, l'instrument de mesure Agilent souffre de plusieurs limitations et inconvénients. La problématique majeur de cette instrument est bien évidement son coût avoisinant les 6/7 k. Cet aspect est l'encontre de notre objectif global et final de permettre à tout développeur de comprendre la consommation énergétique de son application embarquée à bas coût. L'autre aspect non respecté est en rapport avec la flexibilité et l'évolutivité de l'outil. En effet, le format propriétaire de l'outil ne permet aucunement la modification du fonctionnement nominal ou alors l'ajout de nouvelles fonctionnalités pouvant être désirées par le développeur.

Ces raisons et ces limitations nous ont poussées à concevoir une nouvelle plate-forme de mesure énergétique répondant à nos exigences.

5.5.2 Une plate-forme de mesure open source et open hardware

Afin de casser les limitations cités précédemment, nous avons décidé de concevoir à partir de zéro, une plate-forme matérielle de mesure énergétique. Celle-ci est issue d'une conception conjointe matérielle et logicielle. L'objectif de ce schéma est de pouvoir facilement et à faible coût faire évoluer la plate-forme de mesure au gré des besoins rencontrés par les développeurs d'applications embarquées. Ce schéma est basé centralement sur une plate-forme embarquée *Discovery STM32F746NG* [STm17].

Nous décrivons avec plus de précisions, dans la suite de cette section, les deux aspects, matériel et logiciel, de la plate-forme de mesure prototype proposée.

5.5.3 Aspect matériel et mesure électrique

L'état de notre document énonce les différentes approches possibles lors du profilage énergétique d'un objet connecté. Nous avons choisis de baser notre approche sur une méthode matérielle. Ceci implique l'utilisation d'un moyen matériel afin de mesurer la consommation énergétique du système cible. La plate-forme de mesure que nous présentons ici permet justement de jouer ce rôle.

Afin de mesurer la consommation électrique d'un objet, nous proposons la conception d'un circuit matériel dédié à cette tâche. Celui-ci est produit et synthétisé sous forme d'une carte fille directement enfichable dans la plate-forme *Discovery STM32F746NG* représentant l'aspect logiciel de notre conception et aussi le cerveau du processus effectif de mesure électrique.

Le circuit matériel de mesure conçu se base sur une technique traditionnelle de mesure de la tension aux bornes d'une résistance dit *résistance de shunt*. Celle-ci, placée en série avec le système cible à mesurer, est parcouru par un courant proportionnel à l'énergie consommée par le système cible. La tension alimentant le système cible étant fixe, il suffit alors de mesurer l'intensité du courant traversant la résistance afin d'estimer la consommation énergétique du système. La mesure d'un courant n'étant pas chose simple (i.e. il faut se mettre en coupure), la résistance ajouté rentre en jeu et permet de simplifier cette tâche. La différence de potentiel aux bornes de celle-ci est directement liée et promotionnel au courant consommée par le système cible. La valeur d'intensité de ce courant est ainsi aisément retrouvée via la loi d'Ohm en prenant en compte la tension et la valeur de la résistance (i.e. $U = R.i$). Nous choisissons une disposition dite *high side* du circuit de mesure, où la résistance est placée avant le système cible et après l'unité d'alimentation de celui-ci. Les détails par rapport aux avantages de ce choix ont déjà été étudiés dans le chapitre *État de l'art*.

De par la faible valeur de la résistance de shunt utilisée, la différence de potentiel mesurée aux bornes de celle-ci reste faible. Nous utilisons donc un amplificateur opérationnel (AOP) responsable de l'amplification de la tension afin que celle-ci puisse être mesuré avec une faible erreur.

Nous illustrons en Figure 5.4 le fonctionnement de cette approche de mesure. Le courant I_{Load} consommée par le système cible traverse la résistance de shunt R_1 placée en série avec celui-ci. la tension au borne de la résistance est ensuite amplifiée via l'AOP qui fournit en sortie la tension amplifiée V_{out}. Dans notre conception, nous utilisons un amplificateur opérationnel *MAX4378TAUD+* [Int12] alimenté avec une tension de 3.3 V par la plate-forme *Discovery STM32F746NG*. Celui-ci est caractérisé par un gain fixe $G = 20$, ainsi que la possibilité d'amplifier 4 tensions simultanément (i.e. le MAX4378TAUD+ est en fait composé de 4 amplificateurs internes). Ainsi, il est possible de mesurer jusqu'à 4 voies simultanément (i.e. mesurer 4 dispositifs cibles). Le système

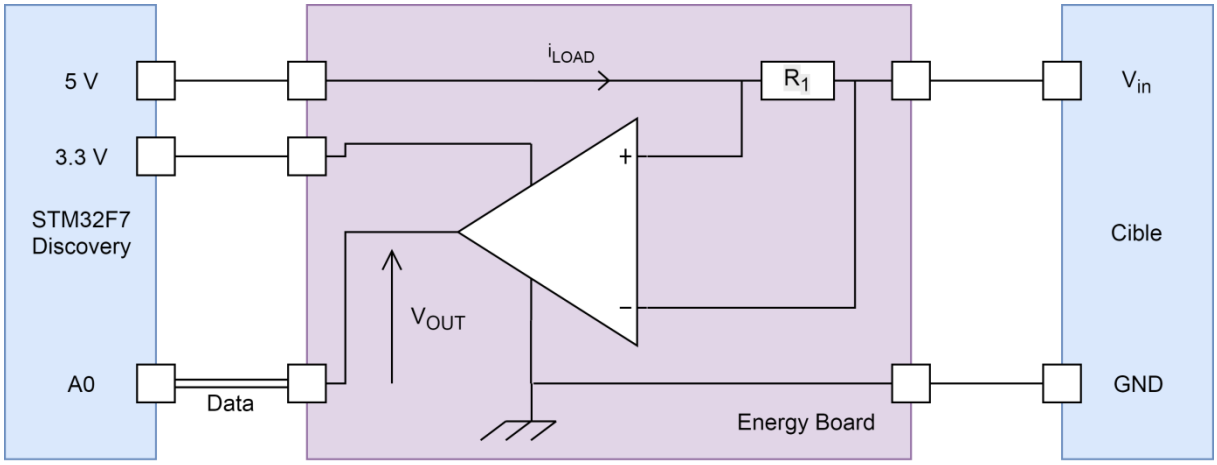


FIGURE 5.4 – Illustration du dispositif de mesure représentant le cœur de la carte fille de mesure.

cible peut être alimenté par une unité séparé ou alors par la plate-forme *Discovery STM32F746NG* directement. De par ce schéma de conception, nous obtenons facilement la valeur de la tension en sortie de l'amplificateur : $V_{out}(t) = GxR_1xi_{LOAD}(t)$.

Le courant consommé par le système cible peut être d'intensité variable. L'utilisation d'une seule résistance avec une unique valeur peut dans certains cas ne pas être suffisante. En effet, la valeur de la résistance de shunt utilisé dans le circuit de mesure définit l'intervalle d'intensité du courant pouvant être mesuré avec une bonne précision et sans arriver à saturation. Afin de résoudre ce problème, nous augmentons notre circuit avec la possibilité de définir différentes résistances pour la voie à mesurer. Nous fournissons 4 emplacements de résistances. Une fois les résistances de shunt placées sur la carte fille de mesure, la sélection de la résistance à utiliser pour la mesure s'effectue manuellement, à l'aide de simples cavaliers.

La tension V_{out} en sortie de l'amplificateur nous permet de retrouver la consommation énergétique du système cible. Pour cela, cette tension doit être quantifiée et échantillonnée. Cette tâche est réalisée grâce à un ADC, qui dans notre conception, est situés sur la plate-forme *Discovery STM32F746NG* qui est responsable de la production de la trace énergétique correspondante à la mesure effectuée. A noter que 3 des tensions de sortie des 4 voies disponibles sont routées sur la carte fille de façon à être directement connectés aux 3 entrées ADC de la plate-forme *Discovery STM32F746NG*.

La carte fille de mesure conçu est totalement illustrés sur ses deux faces dans les Figures 5.5 et 5.6

Nous détaillons donc dans la section suivante l'architecture logicielle utilisée sur la plate-forme *Discovery STM32F746NG*.

5.5.4 Aspect logiciel

Par rapport à la composante logicielle, notre système de mesure est basée sur une plate-forme *Discovery STM32F746NG* [STm17]. Celle-ci est un kit de développement créé et commercialisé par l'entreprise *STMicroelectronics*. D'un coût approximatif de 50, cette plate-forme représente un candidat idéal pour un système

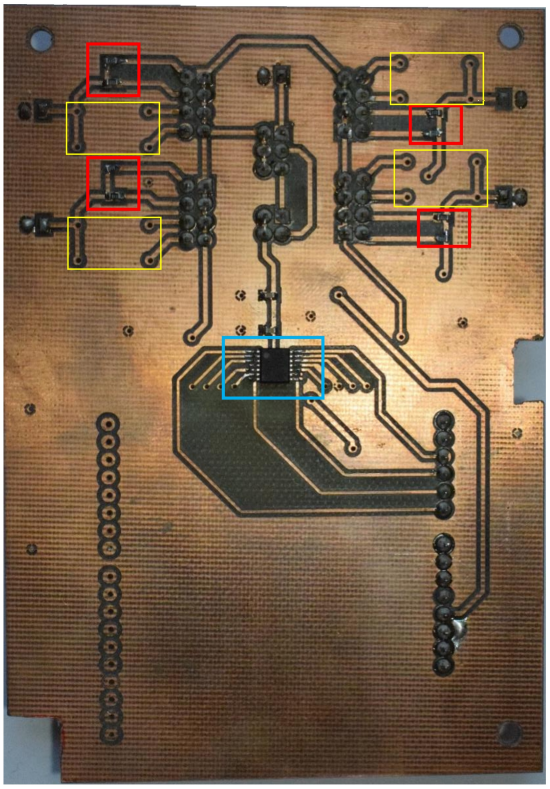


FIGURE 5.5 – Carte fille de mesure de courant. Le coté face illustre : des résistances déjà placées sur chacune des 4 voies (rouge), des emplacements libres pour l'ajout de nouvelles résistances (jaune), l'amplificateur opérationnelle 4 voies (bleu)

de mesure matériel, du point de vue de son prix. Outre le coût effectif, cette plate-forme possède plusieurs caractéristiques dont nous listons ci-dessous, les plus importantes à notre solution.

- Microcontrôleur (MCU) STM32F746NGH6 (architecture ARM, cœur Cortex M7) ;
  - Fréquence maximale : 216 MHz ;
  - 340 Koctets de SRAM (ang. *Synchronous Random Access Memory*) ;
  - 1 Moctet de mémoire FLASH.
- 3 composants ADC 12-bits (Convertisseur Analogique Digital) caractérisé chacun par une vitesse d'échantillonnage maximal de 2 Mega-échantillons/s ;
- Mémoire SDRAM de 128-Mbits (64-Mbits accesible) ;
- Ecran LCD-TFT tactile 4.3 pouces, 480x272 pixels ;
- Connecteur pour carte micro-SD (µSD)/carte MMC (*Multi Media Card*) ;
- Contrôleur Ethernet IEEE-802.3-2002 10/100-Mbit ;
- Contrôleur DMA reliant tous les différents bancs mémoire ;
- Broches GPIO compatibles avec le format Arduino. Ce format est aussi celui de la carte matérielle fille de mesure.

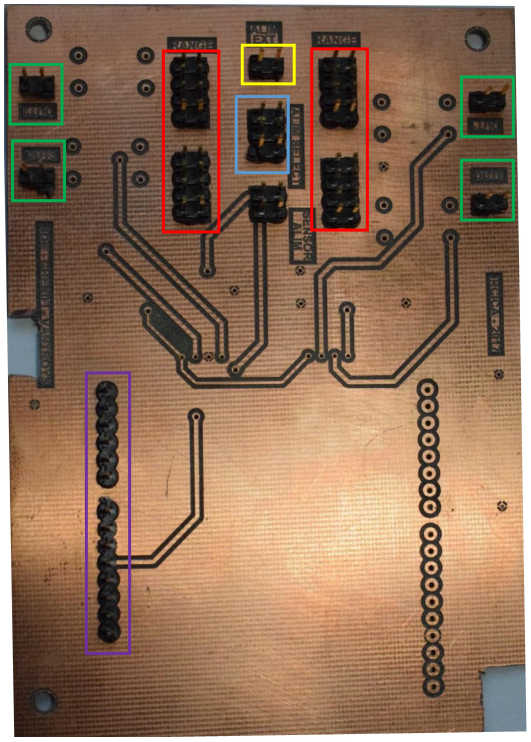


FIGURE 5.6 – Carte fille de mesure de courant. Le coté Pile illustre : les cavaliers permettant de sélectionner les résistances à utiliser sur les voies à mesurer (rouge), les entrées permettant de se mettre en coupure avec le système cible à mesurer (vert), une entrée pour le branchement d'une alimentation externe pour le système cible à mesurer (vert) et le cavalier sélecteur d'alimentation (bleu), les entrés afin d'enficher la carte fille dans la carte *Discovery STM32F746NG* (violet).

5.5.4.1 Objectifs

La plate-forme *Discovery STM32F746NG* représente le cœur et le cerveau de notre plate-forme prototype de mesure. Celle-ci est responsable de l'échantillonnage de la tension de sortie (V_{out}) de la carte fille afin de déterminer la consommation énergétique correspondante. Néanmoins, les fonctionnalités de la plate-forme de mesure vont plus loin. Nous résumons en quelques points ses principaux objectifs et fonctionnalités :

- Mesure en temps réel de la consommation énergétique d'un système cible.
- Minimisation des pertes de données lors de l'échantillonnage de la tension en sortie de la carte fille de mesure.
- Gestion complète du flux de données, jusqu'à la sauvegarde en fichier.
- Affichage d'un graphique de consommation en fonction du temps.
- Réutilisation des données sauvegardées sur un support externe de type carte SD.
- Interaction avec l'utilisateur pour paramétrer et exploiter les données.

Le premier objectif principal peut être résumé en la gestion de la chaîne d'acquisition lors d'une mesure énergétique. Pour cela, la plate-forme échantillonne et quantifie la tension de sortie V_{out} via un ADC intégré. Les

échantillons captés par l'ADC de la plate-forme doivent être mémorisé à la fin de mesure de façon persistante. Cela afin de pouvoir y accéder après coup dans un objectif de visualisation ou traitement. Nous utilisons pour cela un stockage en mémoire SD offert par la plate-forme qui en plus la persistance, offre un moyen de transfert simples des traces énergétiques vers un autre modules ou un PC.

La problématique principal de ce choix d'acquisition et de stockage concerne la faible vitesse d'écriture de la carte SD par la plate-forme. En outre, de par l'architecture des mémoires flash (i.e. le type de mémoire des carte SD), les vitesses d'écritures peuvent être variables d'une écriture à une autre et peuvent suivre un comportement imprédictible en fonction des données à écrire 5.7.

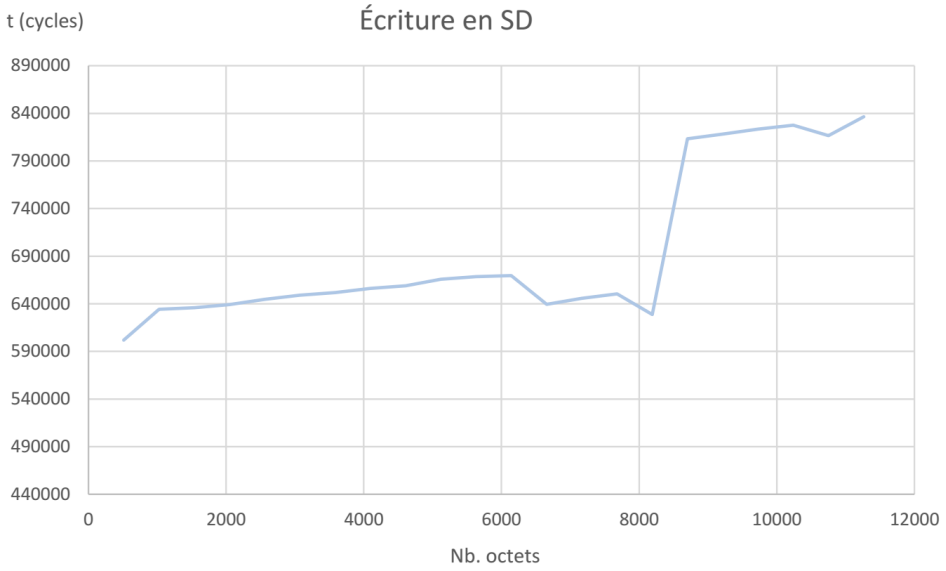


FIGURE 5.7 – Vitesse d'écriture en carte SD en fonction du nombre d'octets à écrire, sur la plate-forme STM32F746NG Discovery.

5.5.4.2 Mémoires d'acquisitions SRAM et SDRAM

Afin de résoudre cet inconvénient posé par la carte SD, nous intégrons dans la chaîne d'acquisition deux type de mémoires déjà présentes sur la plate-forme. La première est la mémoire SRAM représentant la mémoire principale du microcontrôleur. Celle-ci sert de mémoire temporaire pour les échantillons collectés par l'ADC. Un tampon d'une taille fixe est définit dans la SRAM. Une fois ce tampon remplit, les échantillons de l'ADC sont transférés vers une mémoire plus grande, la SDRAM qui servira de tampon final avant la sauvegarde en mémoire SD. Plus précisément, si la durée de la tâche de mesure est indéfinie, celle-ci se terminera lorsque la mémoire SDRAM aura été totalement remplie. Ainsi, Selon la vitesse d'échantillonnage sélectionné (voir point suivant), la durée de la mesure énergétique peut différer grandement. Ce schéma d'acquisition est illustré en Figure 5.8.

Sur notre plate-forme, la SDRAM (*Synchronous Dynamic Random Access Memory*) est un composant externe au MCU. En comparaison de la faible capacité de la mémoire SRAM (i.e. 340 Koctets), la mémoire SDRAM de notre plate-forme offre une capacité utilisable de 8 Moctets. La mémoire SRAM est une mémoire

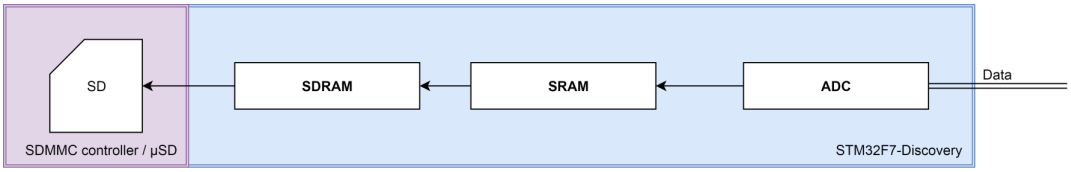


FIGURE 5.8 – Schéma de transfert des échantillons ADC à travers les différentes mémoires de la plate-forme de mesure.

intégrée au MCU. Elle est accessible en un temps équivalent au cycle processeur. La principale interrogation est donc de savoir si la mémoire SDRAM peut assurer des vitesses d'écriture/lecture suffisantes ? Ainsi qu'une prédictibilité de celles-ci ?.

Nous avons effectués l'expérimentation des vitesses d'écriture et de lecture en SDRAM sur notre plate-forme de mesure. Ces expérimentations sont illustrées dans les figures 5.9 et 5.10. Dans un premier temps, nous pouvons observer une nette linéarité dans les courbes de vitesses en fonctions de la taille des données à écrire. Les vitesses moyennes sont quasiment égales aux vitesses moyennes montrant ainsi un faible écart-type. Par rapport aux valeurs de ces vitesses, la SDRAM intégrée à la plate-forme offre respectivement 50 Mo/s et 100 Mo/s pour les opérations de lecture et d'écriture. Ces performances sont largement suffisantes à nos besoins de sauvegarde des échantillons d'ADC, et valide donc l'utilisation de cette mémoire comme tampons final pour le stockage.

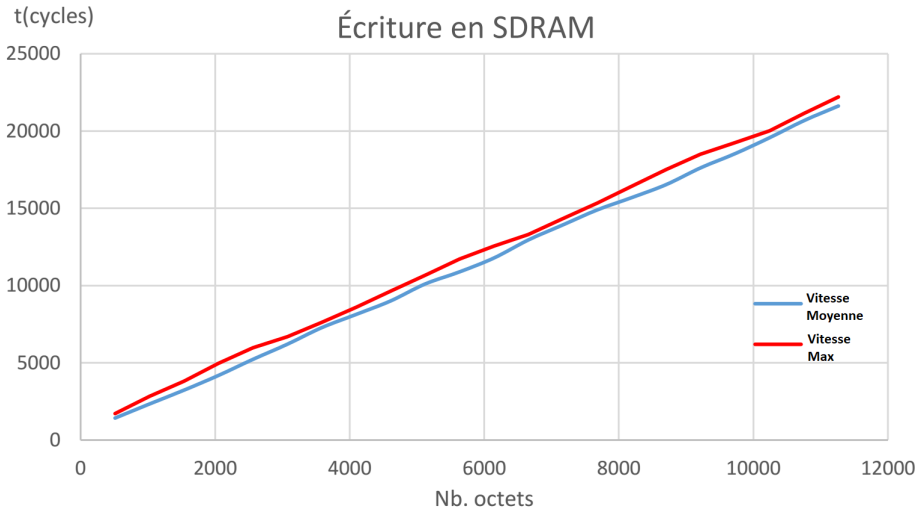


FIGURE 5.9 – Quantification de la vitesse d'écriture en SDRAM en fonction de la taille des données à écrire.

5.5.4.3 ADC

La plate-forme *STM32F746NG Discovery* possède trois modules ADC, qui remplissent le rôle de convertisseur analogique/numérique. Comme nous l'avions énoncé précédemment, de par notre approche de mesure, l'intensité du courant consommé est déduite via la tension en sortie de l'amplificateur opérationnel localisé sur la

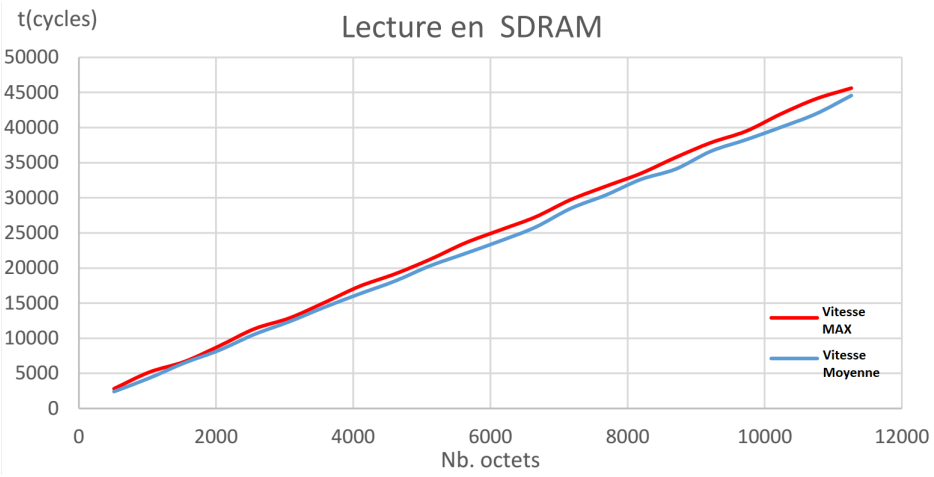


FIGURE 5.10 – Quantification de la vitesse de lecture en SDRAM en fonction de la taille des données à lire.

carte fille de mesure. Afin de pouvoir automatiquement extraire l'intensité du courant, il faut traiter cette tension de sortie par le logiciel de la plate-forme de mesure.

L'ADC rentre donc en jeu afin d'assurer la conversion de la tension du domaine analogique vers le domaine numérique en :

— récupérant à pas régulier (i.e. période) l'amplitude du signal à convertir (échantillonnage).
— puis en convertissant cette amplitude en valeurs compréhensible dans le domaine numérique (quantification).

Les 3 ADC intégrés à la plate-forme de mesure ont chacun une fréquence d'échantillonnage maximale théorique de $f_{max} = 2Mega - echantillons/s$. Cela sachant qu'un échantillon est représenté sur 12 bits. Techniquement, afin de fixer une fréquence d'échantillonnage sur notre plate-forme, il y plusieurs points à prendre en compte :

— L'horloge périphérique du composant ADC. Celle-ci nommée *APB2* est laissée à sa fréquence maximal de $f_{APB2} = 108MHz$ pour des raisons de stabilité d'autre composants aussi cadencées par cette même horloge.
— Un diviseur de fréquence (ang. *prescaler*) $pres_{ADC}$ modifiant la fréquence d'horloge cadençant l'ADC. Ce diviseur peut prendre les valeurs : 2 ; 4 ; 6 ; 8.
— La résolution de l'ADC, res_{ADC} pouvant prendre les valeurs : 6 ; 8 ; 10 ; 12 bits. Pour une résolution optimale nous fixons ce paramètre à 12 bits. Cela voulant dire que la tension d'entrée de l'ADC (de [0-3.3V] sur notre plate-forme) sera quantifiée sur 12 bits (4096 valeurs possibles).
— Le temps de conversion d'un échantillon N_c pouvant prendre les valeurs : 3 ; 15 ; 28 ; 46 ; 84 ; 112 ; 144 ; 480 cycles. Plus le temps de conversion est long, plus l'ADC aura un intervalle suffisant afin de fournir une valeurs numérique précise et représentative du signal échantillonné.

L'équation donnant la fréquence d'échantillonnage par échantillons f_{ADC} de l'ADC est résumée en équation 5.1 :

$$f_{ADC} = (res_{ADC} + N_c) * (\frac{f_{APB2}}{pres_{ADC}}) \tag{5.1}$$

De manière empirique, nous avons observer que pousser les composants ADC intégrés dans notre plate-forme au maximum de leurs possibilités peut générer des artefacts après quantifications. Nous décidons donc de limiter les fréquences d'échantillonnage maximales que peut prendre le composant ADC. Par exemples, en restant sur des fréquences de 27 kilo-échantillons/s ou encore 41 kilo-échantillons/s. Ces fréquences d'échantillonnage, voisines de celles de l'analyseur de puissance AGILENT sont suffisantes à la production de traces énergétiques précises et de confiances. Il est à repréciser, que selon la fréquence d'échantillonnage sélectionnée pour l'ADC, la durée maximal possible pour une mesure diffère. Cela s'explique par le fait que l'acquisition des échantillons s'arrête dans le pire des cas après remplissage total de la mémoire SDRAM. Ainsi, pour une fréquence d'ADC de 27 kilo-échantillons/s (un échantillon étant codé dans logiciel sur 2 octets), la durée maximale d'une mesure est de 2 minute et 30 secondes.

Nous laissons à l'utilisateur de notre plate-forme de mesure le choix d'une fréquence d'échantillonnage à utiliser. Nous présenterons plus tard, l'approche utilisée pour effectuer ce choix.

5.5.4.4 DMA

Dans les points précédent, nous avons montré la méthode d'acquisition des échantillons de mesure. Celle-ci met en scène deux modules mémoires SRAM et SDRAM qui joue respectivement le rôle de tampon temporaire et de tampon final avant le transfert de tout les échantillons de mesure en carte SD. Néanmoins, ce schéma d'acquisition pose un problème de performance au niveau du microcontrôleur. En effet, Les vitesses d'échantillonnage élevés utilisées crées une pression élevé sur le microcontrôleur qui se retrouve à gérer l'acquisition des échantillons du composant ADC, leur mise en mémoire SRAM et leur transfert en mémoire SDRAM. Cette pression et ce goulot d'étranglement cause une perte des échantillons issus du composant ADC et donc une diminution de la fiabilité de la trace énergétique résultante. Pour se prémunir contre ce problème et assurer un fonctionnement fiable de notre plate-forme de mesure malgré une vitesse d'échantillonnage élevé, nous avons choisi de tirer avantage de la DMA (ang. *Direct Access Memory*.

Le DMA est un contrôleur dédié au transfert de données entre mémoires et entre périphériques et mémoires. Ce contrôleur prend quasi-intégralement en charge le transfert mémoire de la source vers la destination. Ce faisant, ce contrôleur allège la charge du MCU, qui peut ainsi exécuter une autre tâche logicielle en parallèle. Le contrôleur DMA présent sur notre plate-forme prototype présente 8 canaux et 8 flux. Toute association flux – canal correspond à une connexion physique entre les deux zones mémoire/périphérique participant au transfert en tant que source/destination.

L'intégration de la DMA dans notre conception logicielle est représentée sur deux grand points :

— Au niveau de l'acquisition des échantillons d'ADC : Chaque échantillon de mesure - une fois prêt - est transféré du registre de stockage de l'ADC vers la mémoire SRAM du microcontrôleur via DMA (schématisé en Figure 5.11). Le stockage des échantillons est réalisé dans un tampon circulaire en SRAM incrémenté automatiquement par le contrôleur DMA.

— Au niveau du transfert vers la SDRAM : le tampon circulaire situé en SRAM est transféré via DMA en SDRAM en deux temps. Une interruption logicielle est enclenchée à chaque moitié de tampon remplie. La moitié en question est copié en SDRAM via DMA. Durant ce même temps, le remplissage de la seconde moitié s'effectue comme décrit dans le point précédent (ADC vers SRAM via DMA).

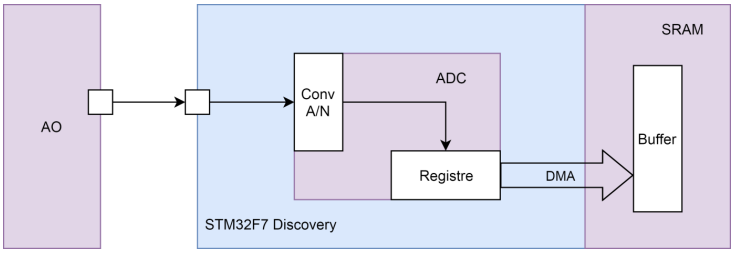


FIGURE 5.11 – Schéma d'acquisition ADC et transfert via DMA en mémoire SRAM.

L'utilisation d'un tampons circulaire subdivisé en deux parties au niveau de la mémoire SRAM ainsi que l'utilisation de la DMA permet à notre plate-forme de réaliser deux transferts mémoires en parallèle. La charge du MCU se retrouve ainsi allégé, celui-ci étant uniquement chargé, lors de la mesure, que du contrôle des interruptions liées aux transferts mémoires entres ADC, SRAM et SDRAM.

5.5.4.5 Stockage SD

Les échantillons de mesures stockés dans le tampons SDRAM sont transférés à la fin de la mesure dans une mémoire de type SDMMC (*Secure Digital Multi Media Card*). La plate-forme *STM32F746NG Discovery* est caractérisé par un port SDMMC pour carte mémoire μSD/MMC. Ce port est interfacé avec le MCU via un bus SDMMC de 4 bits de données. Par défaut, la méthode d'écriture en carte SD par la plate-forme est brut (format RAW). Afin d'apporter une flexibilité et une maintenabilité plus simple, nous avons décidé d'intégrer un système de fichier FAT sur la carte SD. Nous avons choisi de porter l'implémentation logicielle FatFS conçu pour être légère et adéquate aux restriction d'un système embarqué.

Le système de fichier ainsi fonctionnel permet d'assurer une simplicité d'écriture, de lecture et journalisation des traces énergétiques dans la plate-forme de mesure prototype. La trace énergétique est stocké en carte SD sous forme d'un simple fichier de type CSV (Figure 5.12).

5.5.4.6 Interface graphique de la plate-forme

Un des arguments du choix *STM32F746NG Discovery* comme support logiciel pour notre plate-forme prototype est la présence d'un écran tactile intégré. Celui-ci est un écran LCD-TFT tactile de 4.3 pouces, d'une résolution de 480x272 pixels. Cette résolution est suffisante pour les besoins de la mesure ainsi que de sa visualisation. La caractéristique tactile permet d'intégrer des interactions avec l'utilisateur afin de lancer et visualiser la mesure énergétique effectuée. L'objectif est donc de concevoir via cet écran une interface utilisateur graphique tactile de contrôle et de visualisation.

L'interface utilisateur conçu offre deux alternative au développeur embarqué (Figure 5.13) : i) Le lancement d'une mesure et le stockage de la trace associée en mémoire SD ii) L'ouverture et la visualisation d'une

```
1  STM32F7 Discovery energy board application datalog.
2
3  Frequency: 27 KHz
4  Number of samples: 104448
5
6  Sample, intensity(mA)
7  0,73.553113553
8  1,73.479853479
9  2,73.992673992
10 3,74.358974358
11 4,74.725274725
12 5,74.285714285
13 6,74.688644688
14 7,74.835164835
15 8,74.65934065
16 9,74.358974358
```

FIGURE 5.12 – Exemple d'un fichier CSV généré et stocké en carte SD par la plate-forme.

trace précédemment enregistrée. D'un point de vu technique, l'interface utilisateur est développée à l'aide du framework STemWin [STm14] qui fournit une abstraction facilitant la conception d'interface graphique pour une cible embarquée.

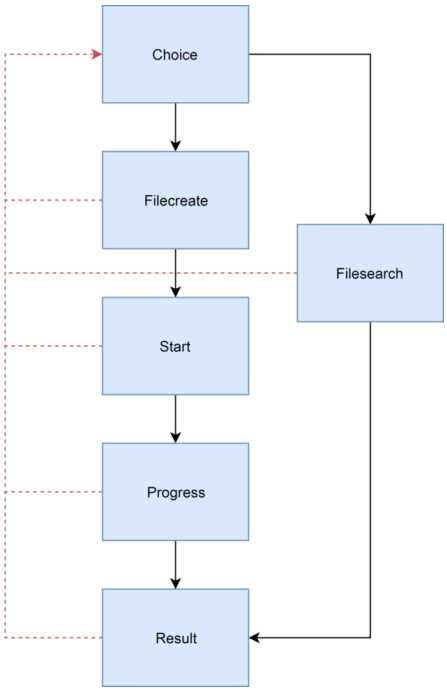


FIGURE 5.13 – Diagramme de séquence des écrans de l'interface utilisateur développée et intégrée dans la plate-forme.

Comme illustré en Figure 5.13 Le premier écran auquel fait face l'utilisateur est un écran de choix le permettant de choisir l'une des opérations possibles (le lancement d'une nouvelle mesure ou la visualisation d'une

ancienne). Cet écran reste assez simple et est constitué de deux boutons permettant de choisir la fonctionnalité désirée 5.14.

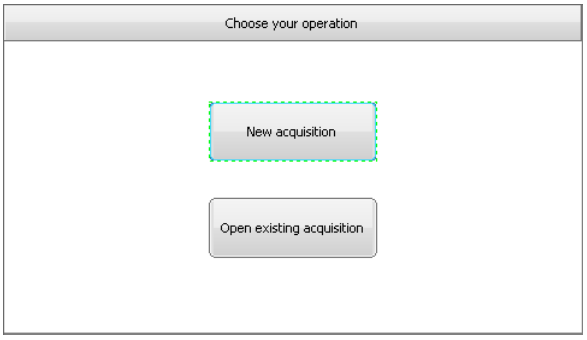


FIGURE 5.14 – Premier écran de l'interface permettant de choisir l'opération à effectuer.

En se concentrant sur la fonctionnalité principal qui est le lancement et paramétrage d'une mesure énergétique, Le second et troisième écran sont des écrans de configuration. Le second écran (Figure 5.15) permet à l'utilisateur de choisir le nom de la trace énergétique qui sera stockée. Un clavier virtuel est affiché à cet escient. Via le troisième écran (Figure 5.16) l'utilisateur (le développeur) peut choisir la vitesse d'échantillonnage de l'ADC ainsi que la durée souhaitée de la mesure énergétique. Le bouton *Start* permet quant à lui de démarrer la mesure effective.

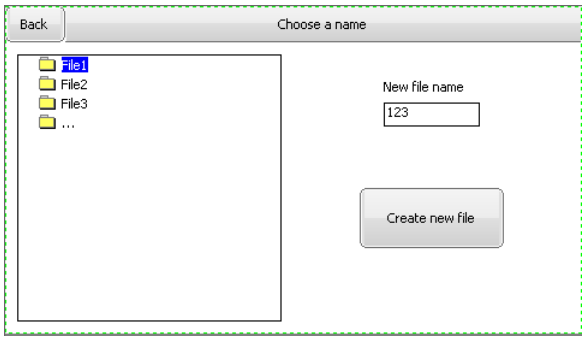


FIGURE 5.15 – Écran de choix du nom du fichier représentant la trace énergétique de la mesure.

Une fois l'opération de mesure enclenché, un écran de progression de type "***barre de progression***" est affiché afin de donner une indication du temps restant pour la complétion de l'opération de mesure. Une fois celle-ci terminée, La trace énergétique représentant la mesure est stockée en mémoire SD et aussi affichée à l'utilisateur de la plate-forme via un écran de résultat (illustré en Figure 5.17 et Fig. 5.18.

L'écran ainsi affiché utilise la notion "d'écran virtuel". Cela permet d'étaler l'affichage de la trace sur plusieurs écrans et avoir par la même voie une visualisation plus précis de la trace (i.e. moins compressé pour l'affichage). La visualisation des écrans virtuels et donc de la totalité de la trace se fait via un curseur glissant (ang. *slider*) situé en bas de l'écran. La caractéristique tactile est aussi utilisé par cet écran de résultat afin de fournir deux fonctionnalités à l'utilisateur : i) Calcul de l'énergie totale de l'acquisition ; ii) calcul de la puissance

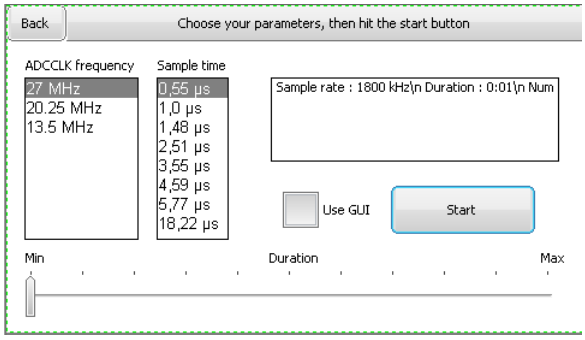


FIGURE 5.16 – Écran de configuration des paramétrés de la mesure énergétique.

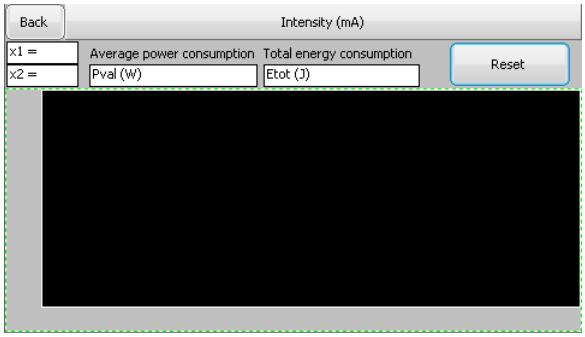


FIGURE 5.17 – Écran de résultat pour l'affichage de la trace énergétique de la mesure venant d'être effectuée.

sur un intervalle choisi par l'utilisateur via un double touché sur l'écran afin de sélectionner les deux points délimitant l'intervalle de calcul. Les résultats correspondants à ces valeurs sont affichés en haut de l'écran.

Enfin, en revenant au début du diagramme de séquence des écran, nous pouvons citer l'existence d'un second choix offert à l'utilisateur de la plate-forme dés le premier écran. Ce choix lui permet de lire et visualiser une trace énergétique déjà présente dans le système (i.e. dans la mémoire/carte SD) (illustré en Figure 5.19. La sélection se fait assez facilement via une fenêtre listant les fichiers de trace énergétique présents dans le système ainsi qu'une fenêtre fournissant quelques informations par rapport au fichier. La trace énergétique une fois ouverte, se visualise et se manipule via l'écran de résultat, de la même façon décrite précédemment pour le cas de la visualisation de la mesure en cours.

5.5.5 Conclusion

Nous avons présenté dans cette section les plates-formes matérielles de mesure utilisées dans le cadre de nos travaux. D'une part, l'analyseur de puissance industriel Agilent, d'une autre la plate-forme maison conçue à partir d'un système STM32F7 et d'une carte fille de mesure. Cette dernière est à l'état de preuve de concept mais représente néanmoins une bonne alternative aux solutions industrielles en terme de coût budgétaire. Nous avons conçue cette plate-forme avec un objectif certes de fiabilité et de performance mais aussi de flexibilité d'utilisation. Cette caractéristique est d'autant plus importante car nous plaçons le développeur au centre des

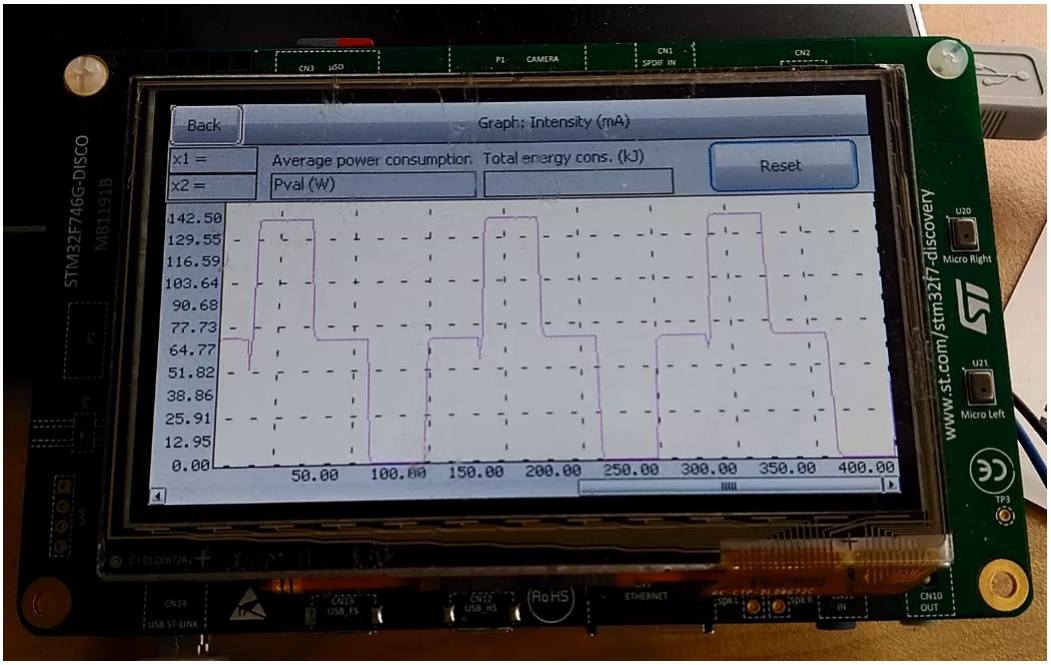


FIGURE 5.18 – Écran de résultat en conditions réelles d'utilisation.

préoccupations. Celui-ci étant généralement freiné dans l'utilisation de plates-formes de mesure énergétique à cause de leur complexité.

5.6 Corrélation de la consommation énergétique et retours visuels

La dernière étape de notre framework de cartographie concerne la corrélation de la consommation énergétique avec l'exécution du code et le retour des informations au développeur. Après l'étape d'instrumentation du code source et celle de mesure énergétique matérielle, cette tâche attache à chaque fonction logicielle, sa consommation énergétique correspondante et la présente au développeur.

Á la fin de la mesure énergétique, le framework possède donc la traces énergétique de la mesure ainsi que les données logiques issues de l'exécution des tags insérés dans le code source. Via ces informations, le framework applique une corrélation synchrone entre les deux flux (i.e données logiques et trace énergétique). Ainsi, cette corrélation représente une superposition directe permettant de tracer l'entrée et la sortie de chaque fonction instrumentée sur la trace énergétique dénotant l'intensité du courant de par le temps. Ayant ce résultat, la consommation énergétique de chaque fonction est calculée en appliquant la formule $Energie = courant * tension_fixe * dure$. Toutes les fonctions instrumentées se voient attachées toutes leurs occurrences, et pour chaque occurrence sa valeur de consommation énergétique ainsi que la fonction appelante. Cela permet de reconstruire un graphe de flot de contrôle énergétique (eCFG) du groupe de fonctions instrumentée de l'application embarquée.

Le framework peut présenter au développeur les résultats du profilage énergétique de l'application de manière brute. De cette façon, le développeur peut observer le profil énergétique de chaque occurrence d'une fonction ainsi

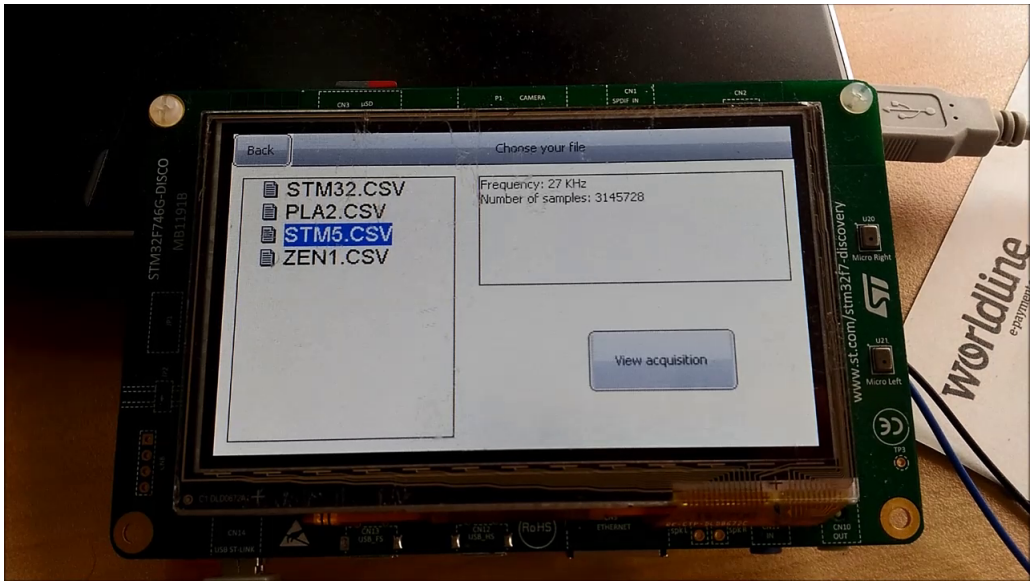


FIGURE 5.19 – Écran d'ouverture d'une ancienne trace énergétique.

que l'arbre d'appel auquel elle appartient. Le framework affiche aussi la courbe de puissance (i.e. intensité du courant) de l'expérimentation horodatée graphiquement avec les entrées et les sorties des fonctions instrumentées.

Néanmoins, afin d'abstraire cette vue des résultats au développeur, nous avons choisi d'ajouter un retour visuel et graphique. Ce deuxième retour fait référence à un désir de présenter une forme plus concise des résultats bruts énergétiques cités précédemment. Ainsi, nous tirons parti du visualiseur open source KCachegrind [Wei] (voir *2.6* en Figure 5.2). Á l'origine celui-ci permet d'afficher et de naviguer dans des profil de performances générés grâce à l'outil Callgrind [Dev]. Le framework l'utilise ici afin de mettre en forme les chiffres brutes et montrer au développeur une vision claire et schématique de l'eCFG. Ainsi via cet outil, chaque fonction instrumentée est affiché avec son coût énergétique, le nombre d'exécution ainsi qu'un lien vers sa fonction appelante. KCacheGrind assigne différentes légendes de couleur en fonction de la consommation énergétique d'une fonction. Cela permet à un développeur de détecter plus facilement les points chauds de consommation énergétique dans le code source de son application embarquée. Nous illustrons l'utilisation de KCachegrind directement dans les exemples d'applications.

5.7 Une instrumentation incrémentale via le traitement des résultats énergétiques ?

L'instrumentation réalisée par le framework permet d'obtenir des données logiques utilisées afin de corréler la consommation énergétique synchrone de l'application embarquée avec son code source interne. Cependant, plus le nombre de fonctions instrumentées est élevé, plus la surcharge causée par cette instrumentation est importante. Même si cette dernière a été décrite comme minime et déductible après-coup de la consommation énergétique globale, il est tout de même toujours profitable de la réduire.

Nous introduisons dans cette optique comme fonctionnalité annexe du framework, une instrumentation incrémentale. L'idée sous-jacente de cette proposition vient des théories du traitement d'image. Particulièrement, celle dictant que l'information importante d'une image se trouve bien plus dans les hautes fréquences (i.e. parties variables de l'image) que dans les basses fréquences (i.e. parties stables de l'image). Appliqué à notre problématique, cela équivaut à poser l'hypothèse selon que les fonctions associées à un comportement énergétique très variable sur la trace énergétique produite, sont des fonctions pouvant effectuer plusieurs traitements différents. Ainsi, l'instrumentation des sous-fonctions appelées par ces dernières est plus profitable pour en apprendre plus sur le comportement énergétique de l'application que l'instrumentation des autres fonctions caractérisées par un comportement énergétique stable.

Pour réaliser ce processus, nous utilisons une détection des pics sur chaque sous-trace énergétique attachée à chaque fonction instrumentée. Pour cela, nous réalisons une simple dérivée des sous-traces énergétique afin de mettre en évidence des changements brusques. Puis, nous comptons le nombre de pics dépassant un certain seuil énergétique (le développeur peut fixer ce seuil librement). Si ce nombre est significatif (selon le nombre de pics minimal défini par le développeur), alors les sous-fonctions de la fonction courante sons elle aussi instrumentées et une nouvelle mesure énergétique est engagée afin de fournir au développeur plus d'informations quant à quelle sous-fonction incombe telle ou telle comportement brusque.

5.8 Exemples d'applications

Dans cette section, nous illustrons l'utilisation du framework de profilage énergétique sur deux exemples d'applications. Le premier est une application embarquée de stress. Celle-ci effectue en séquence différents tests de stress assez variés en terme de domaine. Le second exemple quant à lui correspond à une application embarquée réelle réalisée au niveau de Worldine *Connected Kitchen*.

5.8.1 Une simple application embarquée de stress

L'exemple mis en évidence ici est une simple application embarqué de stress. Plus précisément, celle-ci effectue différentes exécutions de divers tests que nous pouvons lister comme suit : Allumage de Leds RGB, Un tri à bulle, une multiplication de matrice, un chiffrement et déchiffrement AES, un effacement de plusieurs blocs en mémoire flash et enfin l'utilisation d'un servomoteur connecté en tant que périphérique externe de la plate-forme embarquée. Cette plate-forme est un système embarqué STM32F4-Discovery [Stm] basé sur un microcontrôleur ARM Cortex M4 32-bit. Elle est alimentée via une tension fixe de 3.3 V.

Une fois l'instrumentation des fonctions effectuées, la mesure énergétique effective peut se faire. À la fin de la mesure, le frameworks possède donc les données logiques issues de la plate-forme embarqué ainsi que la trace énergétique produite par la mesure. La superposition des deux flux et de leur corrélation temporelle directe permet d'obtenir la consommation de chaque fonction logicielle instrumentée. Une illustration de ce processus est décrit par la figure 5.20.

Sur cette illustration, nous pouvons voir la courbe de consommation énergétique de l'application de test. Plus précisément, la consommation en courant (mA) en fonction du temps (i.e. La tension est fixe). Sur cette courbes, sont ajoutés plusieurs lignes verticales qui correspondent aux différents début et fin de fonction logicielle. Par

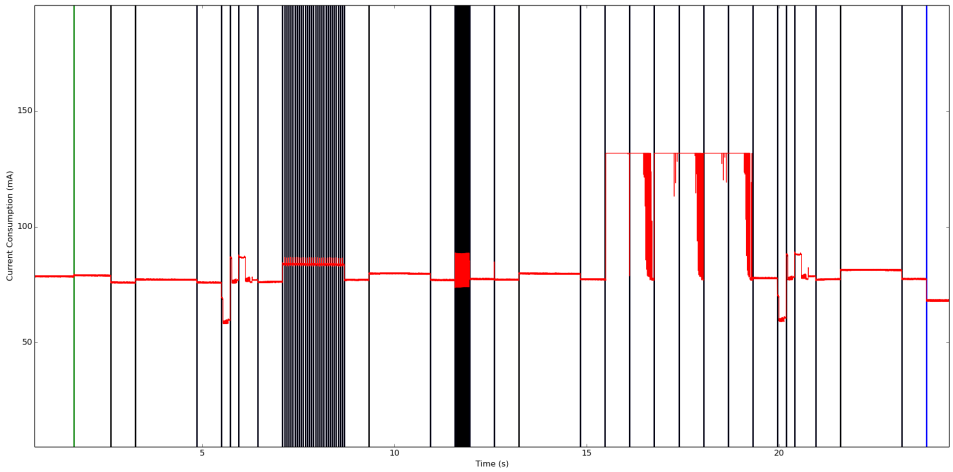


FIGURE 5.20 – Trace énergétique produite après mesure et profilage de l'application de test.

ailleurs, nous nous rendons compte aisément que la grande majorité des lignes verticales sont positionnés au début et fin de segments ayant une forme qui dénote par rapport aux segments voisins. Cela s'explique directement par le fait que n'importe quel traitement différent effectué par le système va produire une trace énergétique différente. C'est l'un des points les plus important nous ayant poussé à étudier les traces énergétiques effectives produites par un vrai instrument de mesure. Car tout comportement logique d'un système peut être observé au niveau physique, sur son comportement énergétique.

Le framework nous délivre aussi une autre vue d'ensemble du résultat du profilage. Celle-ci est généré via l'utilisation de Kcachegrind auquel on fournit le graphe dynamique d'appel de fonction ainsi que la consommation énergétique de chaque fonction. Sur cette vue illustrée en figure 5.21, nous pouvons observer un des affichages (certainement le plus important) offert par l'outil Kcachegrind. Sur la partie inférieur, l'outil nous offre donc une vue de l'arbre dynamique d'appel de fonction.

Sur la partie haute, nous avons un affichage en bloc qui se concentre sur la consommation de chaque fonction. Ainsi la consommation de chaque bloc est proportionnelle à la superficie de celui-ci. Les résultats énergétiques numériques sont données en mJ (millijoule) et le nombre d'appels de la fonction en question est notée au-dessous. Les inclusions entre blocs illustrent d'une façons différente l'arbre dynamique d'appel de fonctions.

Dans l'exemple illustré, nous pouvons voir que la fonction utilisant le servomoteur (*servo_move()*) affiche une consommation énergétique élevée. Il en va quasiment de même pour la fonction d'efficacement de secteur dans la mémoire flash (*FLASH_EraseSector()*). La fonction de stress *bubble_sort()* possède aussi une consommation énergétique significative. Néanmoins cela est liée avec le nombre de fois que le stress est reproduit. La fonction *BubbleSort()* est ainsi appelée 30 fois !

Enfin nous pouvons remarquer que la fonction d'attente active *wms_delay()* dénote une haute consommation. Cela s'explique par des longs temps d'attente insérés dans le programme. Une attente est réalisée entre chaque

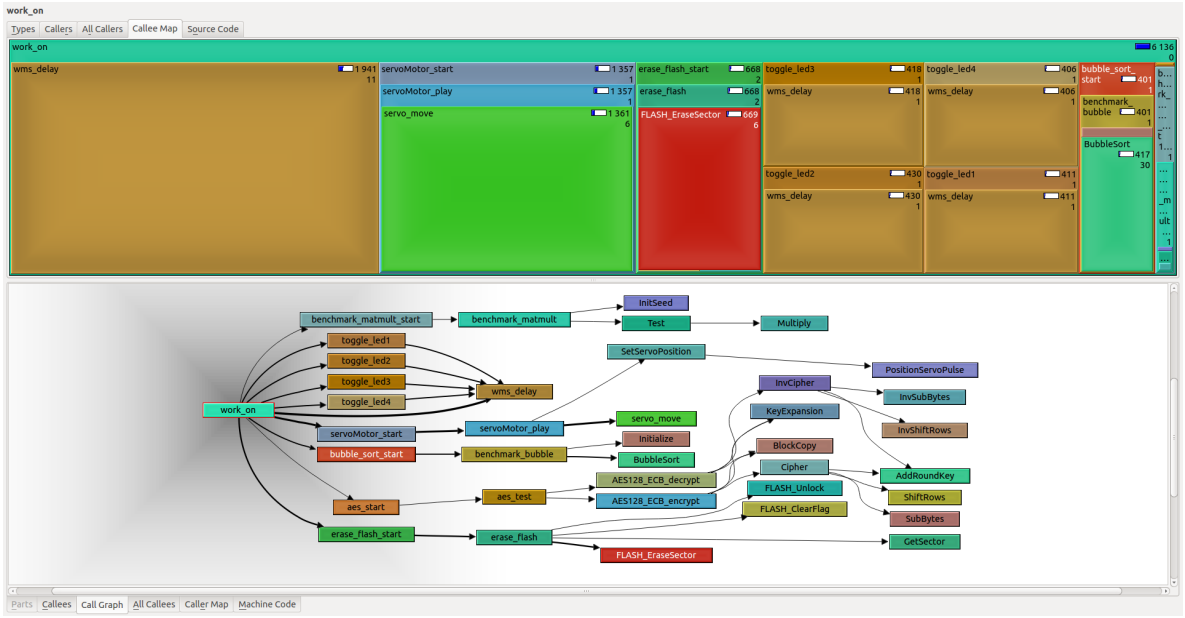


FIGURE 5.21 – Aperçu d'un affichage produit par le visualiseur Kcachegrind après une passe d'un profilage énergétique du framework de cartographie.

opération de stress distinctes. Puis, une attente est fait lors des fonctions utilisant les LED (*toggle_ledx()*) entre l'allumage et l'extinction de celles-ci.

5.8.2 Connected Kitchen

Connected Kitchen est un objet connecté développé par la société Worldline partenaire de cette thèse. Cette objet se présente sous la forme d'un magnet [4] (cf. Figure [**?**]) connecté au réseau local de la maison. La fonction principale de cet objet est de faciliter l'ajout d'articles à la liste de courses du foyer. Les utilisateurs peuvent ajouter des produits de tous les jours à leur liste de courses en les scannant ou en utilisant la commande vocale intégrée à l'appareil [5].

L'appareil est composé d'un lecteur capable de lire les code-barres des produits scannés. L'objet connecté envoie les références des produits scannés aux services *Cloud* de Worldline. De là, l'infrastructure interne distante met à jour la liste de courses en y ajoutant les nouveaux produits scannés. L'utilisateur peut consulter cette liste, via l'application mobile du détaillant ou son site Web.

La *Connected Kitchen* intègre un module WiFi lui permettant de se connecter sur Internet afin de communiquer avec les services IT de Worldline. Coté processeur, elle est basée sur un microcontrôleur STM32F103ZE basé donc sur un ARM Cortex-M3 cadencé à 72 MHz avec 512 Koctets de mémoire Flash et 64 Koctets de mémoire SRAM. Elle est alimentée via une batterie LIPO [3.7 - 4] V d'une capacité de 320 mAh. La recharge de la batterie s'effectue via un simple câble USB. Du fait de la faible capacité de cette dernière, la consommation

4. Magnet : objet à aimant collable sur un frigo.
5. La commande vocale est une fonction encore en test.

FIGURE 5.22 – La *Connected Kitchen* conçue par Worldline.

énergétique de la *Connected Kitchen* prend une place importante dans le développement du logiciel embarqué. Nous pensons donc que le profilage énergétique des fonctions principales de l'objet représente un excellent exemple d'application.

Nous décrivons dans ce qui suit les résultats du profilage énergétique de trois opérations distinctes et primordiales pouvant être effectuées par la *Connected Kitchen*. Celles-ci sont :

— l'ajout d'un ou plusieurs produits à la liste de courses via la lecture du code-barres associé ;
— l'ajout d'un ou plusieurs produits à la liste de courses via la commande vocale intégrée ;
— l'appairage de l'objet au réseau local de la maison de l'utilisateur.

Lecture de code-barres

Le service principal rendu par la *Connected Kitchen* réside dans la simplification de l'élaboration de la liste de course, en mettant en avant un ajout continu des produits à celle-ci. La façon la plus simple d'ajouter un produit est l'utilisation de la fonction de lecture de code-barres.

Dans un objectif d'expérience utilisateur, l'utilisation de la lecture de code-barres sur la *Connected Kitchen* se veut très simple. L'utilisateur prend en main l'objet connecté, puis enfonce le grand bouton afin de lancer le lecteur de code-barres. En gardant le bouton enfoncé l'utilisateur peut alors faire plusieurs lectures de différents produits. Une fois le bouton relâché, l'utilisateur peut encore ré-appuyer afin de continuer la lecture de nouveaux code-barres. Après un délai d'inactivité de 3 secondes, les code-barres lus et enregistrés sont envoyés via le WiFi au service IT distant (Cloud) pour traitement et ajout effectif des produits à la liste de course.

Nous proposons de profiler ce cas d'utilisation de notre objet connecté. Nous illustrons dans la figure 5.23 la trace énergétique résultante de la capture faites lors de ce scénario en précisant la lecture de quatre code-barres différents. Nous pouvons voir que de la même façon que le précédent profilage, plusieurs changements de

consommation énergétique se produisent lors de l'appel de différentes fonctions logicielles. En comparaison avec le précédent profilage, la consommation instantanée en courant est ici bien plus élevée avec ce nouvel objet connecté.

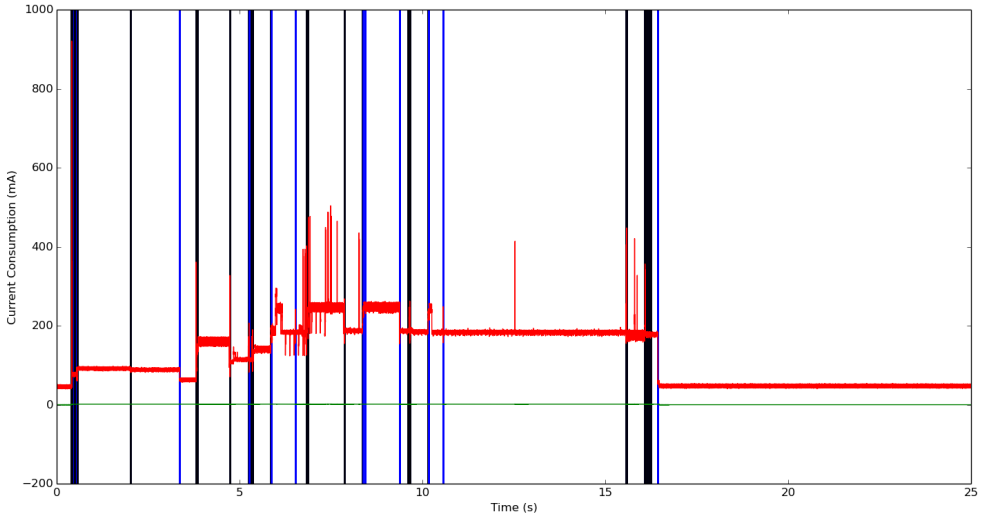


FIGURE 5.23 – Trace énergétique capturée après lecture de quatre code-barres via la *Connected Kitchen*.

Nous illustrons ci-dessous la visualisation Kcachegrind issue du profilage. La figure 5.24 montre un affichage par blocs des fonctions profilées. La consommation énergétique est ici affichée sous une deuxième forme qui représente un pourcentage de contribution à la consommation énergétique totale. La partie droite de l'illustration par blocs celui-ci présente une chaîne isolée d'exécution de fonctions liée avec l'étape d'initialisation du logiciel (*main_setup_phase()*). Dans cette étape, les composants périphériques sont entre autres initialisés pour pouvoir être utilisés lors du fonctionnement continu de l'objet.

Le graphe d'appel de fonctions lié à cette étape d'initialisation est illustré à la figure 5.25. Ce graphe permet ici au développeur d'analyser la consommation énergétique issue de l'initialisation des différents composants périphériques nécessaires (moniteur de batterie, mémoire SDRAM, mémoire SD, RTC, WiFi). L'initialisation de la RTC [6] (*start_RTC()*) représente d'ailleurs la plus grosse part dans la consommation énergétique de l'étape d'initialisation.

L'étape d'initialisation décrite est commune quel que soit le scénario effectué, en comparaison avec le fonctionnement continu de la *Connected Kitchen* (fonction *main_event_loop()*). Nous décrivons cette deuxième étape à la figure 5.26. En analysant conjointement cet affichage et celui illustré en Figure 5.24, un développeur peut faire plusieurs constatations.

La fonction liée avec la lecture de code-barres (*scan_button_push()*) contribue en grande partie à la consommation énergétique du scénario, en représentant 44% de la consommation de la fonction continue de traitement

6. RTC : Real Time Counter

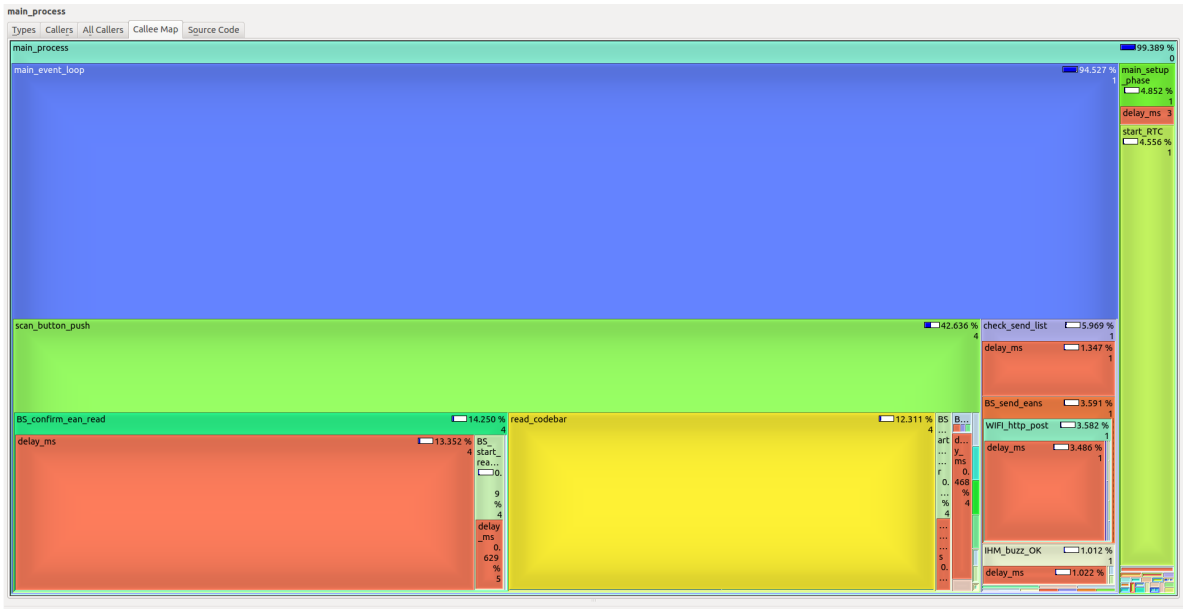


FIGURE 5.24 – Affichage Kcachegrind avec une vue par blocs du profilage de la lecture des code-barres.

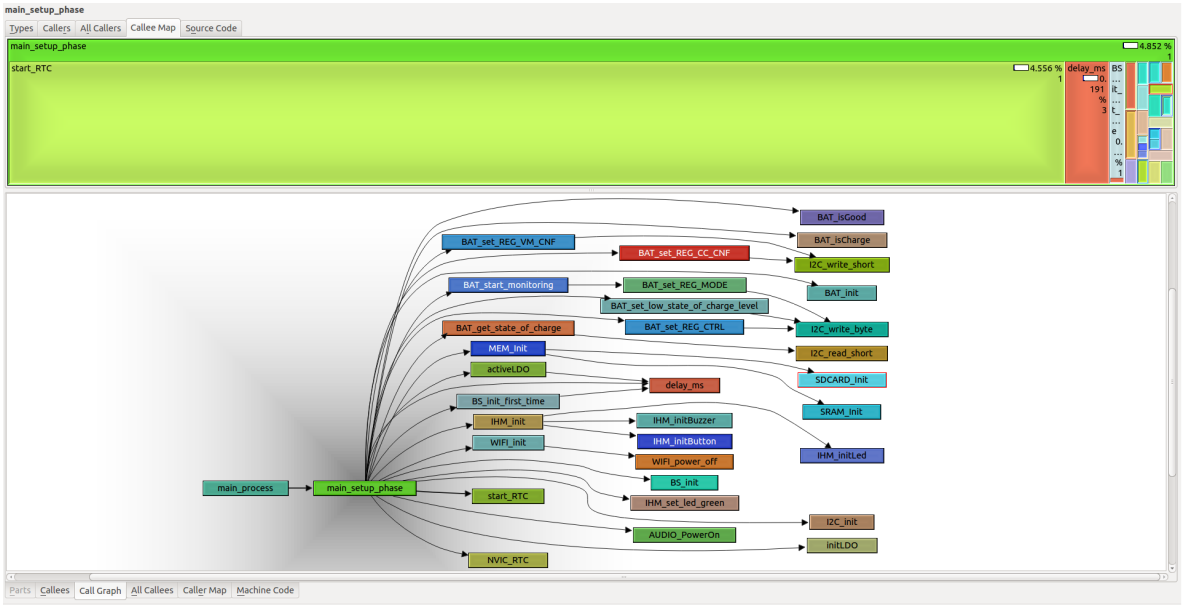


FIGURE 5.25 – Graphe d'appel lié à l'étape d'initialisation de la *Connected Kitchen*.

des événements *main_event_loop()*. Celle-ci regroupe deux composantes principales en terme de consommation énergétique. La première est le scan effectif avec la fonction *read_codebar()*. La deuxième est une fonction de notification, de délai et de réinitialisation du scanner entre chaque lecture de code-barres (fonction *BS_confirm_ean_read()*).

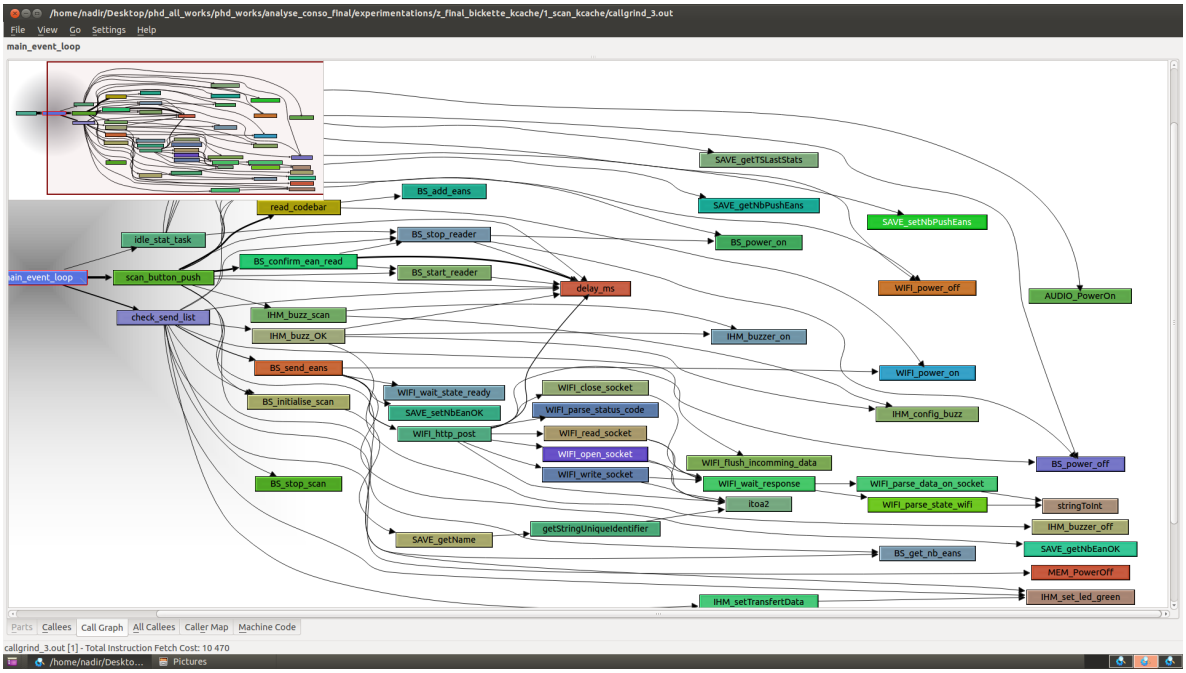


FIGURE 5.26 – Graphe d'appel lié à au fonctionnement continu de la *Connected Kitchen* lors de la lecture de code-barres.

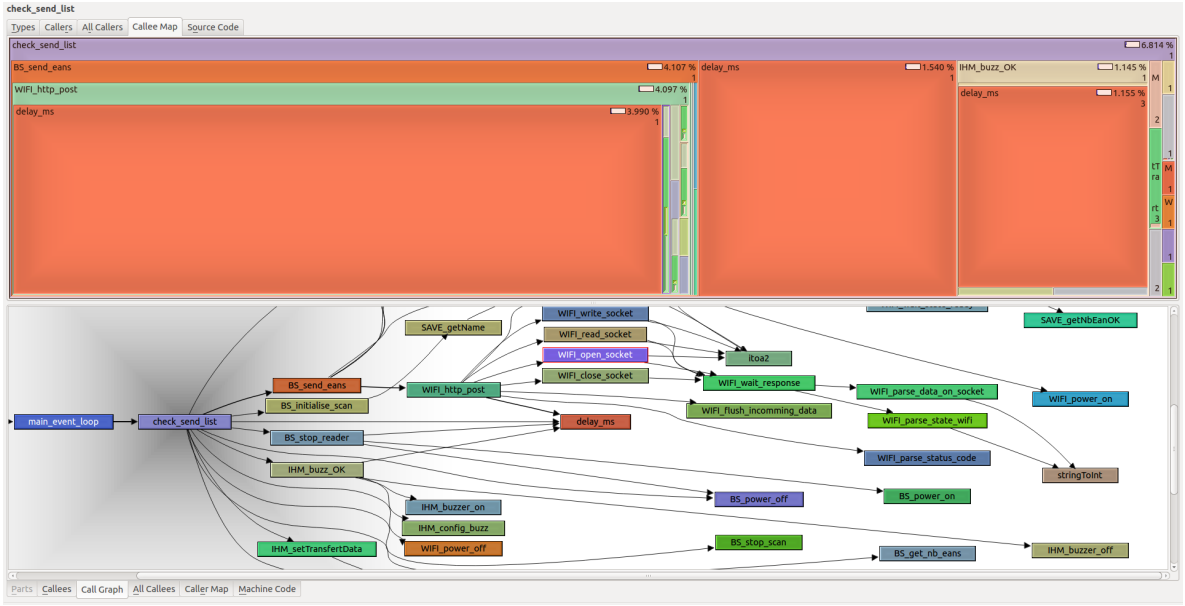


FIGURE 5.27 – Zoom sur la fonction d'envoi de code-barres via WiFi de la *Connected Kitchen*.

Concernant l'envoi des code-barres lus en WiFi vers les service distants de Worldline, c'est la fonction *check_send_list()* qui s'en charge. Celle-ci contribue à hauteur de 6% à la consommation globale. Nous donnons

un zoom de cette fonction dans la figure 5.27. Cette fonction effectue une vérification du nombre de code-barres lus ainsi que leur envoi. Il est tout de même à noter que la plus grande part de la consommation ici est représenté par l'envoi effectif. La fonction *BS_send_eans* ainsi que sa sous-fonction *WIFI_http_post* sont responsables de l'envoi par WiFi. Une grande partie de cette consommation énergétique ne réside néanmoins pas dans l'utilisation effective du WiFi mais dans une attente active de par la fonction *delay_ms*. Cette attente est nécessaire pour surveiller la réception via le module. Une amélioration possible ici est donc l'utilisation d'un module WiFi acceptant le réveil sur interruption lors de la réception.

Commande vocale

L'objet *Connected Kitchen* peut être utilisé d'une seconde façon afin d'ajouter des produits du quotidien à la liste de course de l'utilisateur. En effet, une commande vocale est intégrée sur l'objet pour utiliser sa propre voix. En terme de scénario d'utilisation, cela se résume ainsi : l'utilisateur utilise la *Connected Kitchen* et enfonce le petit bouton bleu situé juste au dessus de celui utilisé pour la lecture de code-barres. Tant que ce bouton est enfoncé l'utilisateur peut énoncer différent produits (accessoirement la quantité aussi) qui seront ajoutés à la liste de course. Il est important de souligner que la reconnaissance vocale n'est pas faites sur l'objet mais sur un serveur distant. L'objet enregistre donc la phrase énoncée et envoie cet enregistrement au format *wave* [7] pour interprétation.

Sur un plan technique, les composants qui rentrent en jeu sont le microphone, l'utilisation d'une mémoire SDRAM comme extension de la mémoire SRAM du microcontrôleur, une carte SD permettant la sauvegarde finale de l'échantillon vocal capturé et enfin le module WiFi pour l'envoi de ces échantillons vers le serveur distant.

Nous décrivons ci-dessous le résultat du profilage énergétique d'un scénario d'utilisation de la commande vocale où trois produits sont énoncés. La figure 5.28 illustre le résultat du profilage énergétique avec un affichage par bloc. Nous montrons ici uniquement la fonction continue de traitement des événements *main_event_loop()*. L'étape d'initialisation est bien-sûr toujours présente, néanmoins elle reste identique à ce qui a été vu précédemment. Nous illustrons aussi une partie du graphe d'appel de fonctions en figure 5.29.

Nous remarquons dans cette illustration, la fonction *speak_button_push()* permettant de gérer l'appui sur le bouton de commande vocale ainsi que l'enregistrement temporaire de l'échantillon. Celle-ci représente quasiment 12% de la consommation énergétique totale. Cela est due à l'utilisation active du microphone pour la capture vocale des produits énoncés. la fonction *check_send_list()* est aussi présente dans ce scénario avec une large contribution (54%) à la consommation énergétique globale. Le but de cette fonction est de vérifier s'il y a des échantillons à envoyer. En présence de captures vocales, une copie de l'échantillon est effectuée vers la mémoire persistance représenté par la carte SD. Cette opération est soutenue par la fonction *SRAM_Audio_To_SDCARD()* qui représente 6.8% de la consommation globale. Cette consommation est difficilement compressible en comparaison avec l'autre composante principale représentée par la fonction *WS_Put_a_Vocal()*.

Cette composante représente 18.7% de la consommation globale. La majeure partie de cette consommation est due à la fonction *WIFI_write_socket_binary()* qui se charge de l'envoi effectif des échantillons vocaux en WiFi. Néanmoins, la moitié de la charge énergétique, illustrée ici par la fonction *WIFI_wait_response()* pourrait

7. WAV : Waveform Audio File Format.

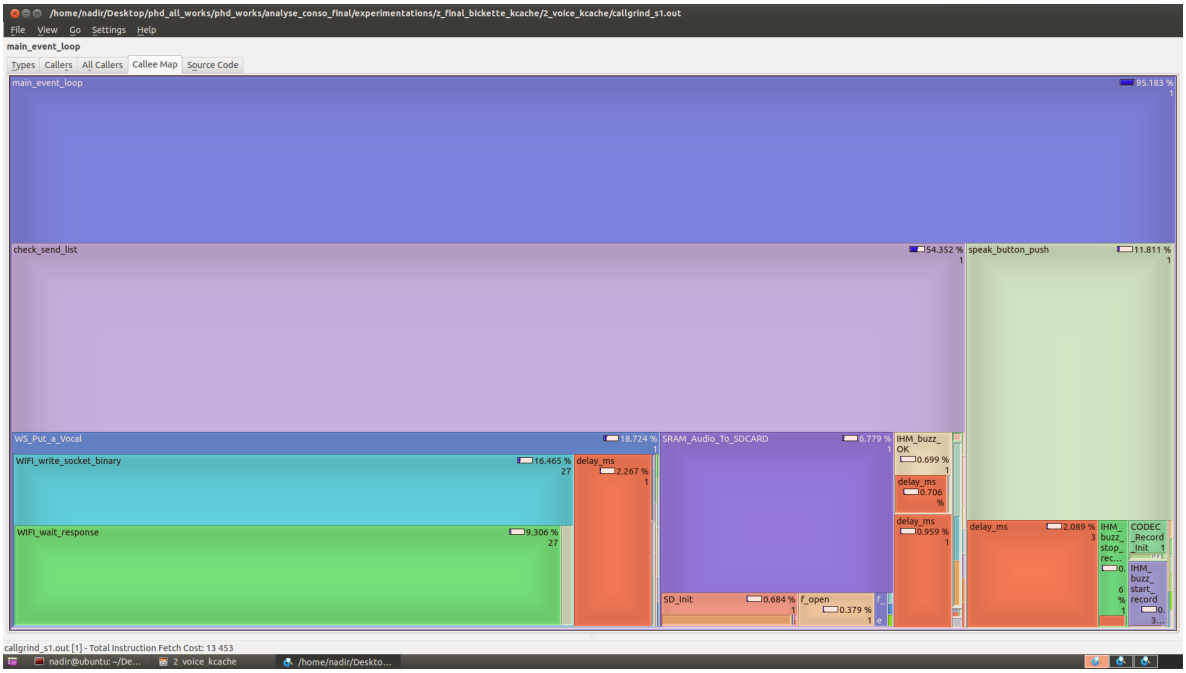


FIGURE 5.28 – Affichage Kcachegrind avec une vue par blocs du profilage énergétique de l'utilisation de la commande vocale.

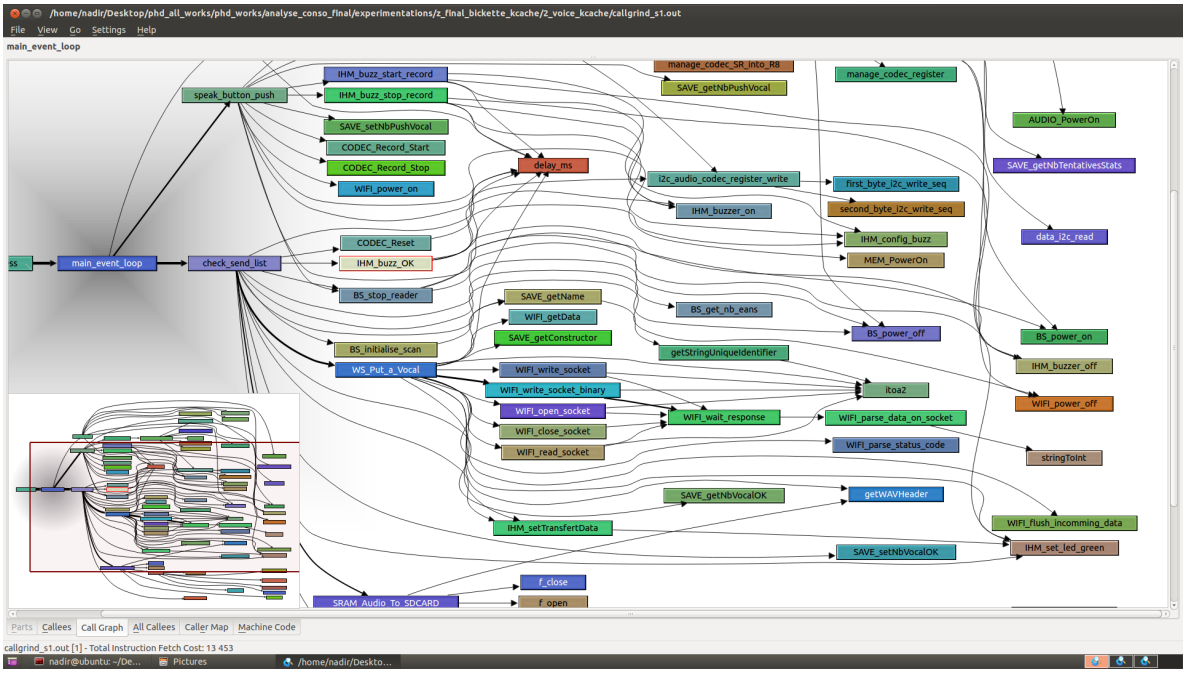


FIGURE 5.29 – Arbre d'appel de fonctions - en partie - du profilage énergétique de l'utilisation de la commande vocale.

être ici atténuée en apportant des modifications à la façon d'attendre la réponse du serveur. En effet, une attente active est effectuée afin de recevoir la réponse du serveur et la vérifier. Un fonctionnement via interruption pourrait donc grandement améliorer l'efficacité énergétique ici.

Appairage au réseau local

ToDo : BOF, pas forcément utile d'en rajouter avec déjà tous les screens avant et vu que l'appairage n'est fait qu'une seule fois et que cela n'est pas forcement utile et impactant quand on parle de consommation ? ? !

5.9 Synthèse et conclusion

Nous avons décrit dans ce chapitre un nouveau cycle de développement d'applications embarquées mettant la consommation énergétique au centre des préoccupations. Nous nous sommes basés sur ce cycle afin de concevoir un framework de cartographie liant la consommation énergétique de l'application au code source de celle-ci au niveau fonctionnel. Ce framework repose sur une méthode de mesure énergétique expérimentale matérielle dans le but d'obtenir les chiffres énergétiques les plus précis et proche de la réalité.

Globalement, les travaux effectués dans ce chapitre ont pour objectif principal le support et l'aide au développeur lors du profilage énergétique d'un code embarqué, phase qui peut être complexe surtout en présence d'une platine matérielle de mesure. Dans ce cadre, nous avons incorporé à notre framework une instrumentation automatique du code source. Puis, nous avons conçu une plate-forme matérielle de mesure énergétique bas-coût et flexible pouvant guider et faciliter la tâche du développeur lors de l'étape de profilage.

L'approche a été appliquée et expérimentée sur plusieurs applications embarquées donnant des résultats concluant en terme de précision de profilage et d'aide au développeur. Ces expérimentations ont été décrites dans la section précédente notamment avec le profilage énergétique du code source d'un vrai produit industriel, la *Connected Kitchen*.

Cependant, notre proposition de framework de profilage énergétique n'est, jusqu'à présent, réellement précise qu'avec le profilage de consommation énergétique synchrone. En d'autres termes, lorsque l'énergie est consommée de façon synchrone à l'exécution effective de la fonction qui en est responsable dans le code source. Malheureusement, en ce qui concerne plusieurs périphériques externes, Cette asynchronisme prend vraiment du sens lorsqu'on aborde les composants périphériques embarqués sur la plate-forme cible, où la consommation énergétique peut être asynchrone contrairement à celle du processeur par exemple.

Ce frein est parmi les nombreuses raisons qui nous ont poussés à étudier de plus près la consommation énergétique des composants périphériques embarqués. Nous détaillons cette étude dans le chapitre suivant.

Chapitre 6

Inférence d'automates énergétiques de périphériques embarqués

Nous détaillons dans ce chapitre, notre proposition d'un processus d'inférence de modèles énergétiques pour systèmes embarqués. Précisément celui-ci se concentre sur la génération de modèles énergétiques résumant le comportement énergétique des périphériques embarqués. Nous décrivons ci-dessous les raisons et motivations qui nous ont poussés à cette proposition en détaillant la construction de celle-ci.

6.1 Introduction

À travers le chapitre d'étude sur la consommation énergétique des modules cellulaires, nous avons pu nous rendre compte de plusieurs faits importants. Dans un premiers temps, nous avons remarqué que le comportement des modules cellulaires peuvent être assez complexes. Ces derniers étant dirigés par une machine à états qui est elle-même pilotée par différents paramètres non fixés par le module cellulaire mais par l'opérateur du réseau. Nous pouvons citer, entre autres, le taux de remplissage du tampon d'émission qui enclenche - ou non - une transition vers un état de consommation supérieur. L'opérateur peut donc modifier n'importe quel paramètre de la machine à états d'un modem selon plusieurs facteurs qui lui sont propres (i.e. fonctionnement de son réseau de cœur, surcharge des stations de base BTS, etc.). Sachant cela, nous pouvons déclarer que le comportement d'un module cellulaire peut être modifié à maintes reprises durant toute la durée de vie de celui-ci.

Outre le comportement variable des modules de communication cellulaire, notre raisonnement a aussi été nourri par le monde des systèmes embarqués tel qu'il se présente à nous aujourd'hui. En effet, le boom de l'Internet des Objets a été accompagné par la démocratisation de différents petits périphériques embarqués. Ces composants peuvent globalement être classés selon trois catégories : capteurs, actuateurs, modules de communications (filaire ou sans-fil). Pour chacune de ces classes, plusieurs modèles de périphériques ont été conçu afin de remplir différentes fonctions (e.g. divers phénomènes physiques à écouter) ou alors la même fonction de différentes manières (e.g. transmettre/recevoir des données via différentes technologies de communication). À leur tours, chaque périphérique est caractérisé par une architecture de conception matérielle ainsi qu'une interface

logicielle qui lui sont propres. Au final, il est aisé de retrouver deux ou même plusieurs composants périphériques embarqués qui remplissent la même fonction mais qui différent dans le comportement énergétique sous-jacent.

Une développeur embarqué se retrouve donc très souvent à tester, expérimenter puis intégrer différents périphériques embarqués voir en remplacer certain durant la phase de développement. Dans un contexte et environnent contraint en énergie, le développeur se retrouve souvent ainsi à réapprendre le comportement énergétique de chaque nouveau périphérique afin de pouvoir livrer une application embarquée efficace en énergie. Cette étape de compréhension est assez fastidieuse et longue, néanmoins celle-ci est primordiale afin de permettre au développeur d'être proactif.

Nous nous proposons ici, d'aider le développeur dans ce processus en lui offrant un support pour la compréhension du comportement énergétique des composants périphériques. En gage de support, nous proposons de fournir au développeur une méthode d'inférence de modèles énergétiques pour composants périphériques embarqués. Les modèles énergétique sont directement liés avec l'API (Interface Applicative de programmation) logicielle du pilote contrôlant chaque activité du périphérique. Ce schéma, très proche du logiciel manipulé par le développeur, accorde à ce dernier une compréhension plus simple et direct du comportement énergétique d'un périphérique donné.

6.2 Objectif et Motivation

Notre principale motivation et objectif de par notre proposition décrite dans ce chapitre est de fournir au développeur embarqué un moyen d'être proactif lors de la conception d'une application logicielle embarquée ayant des contraintes énergétiques. Nous proposons pour cela un processus d'inférence de modèles énergétiques pour des composants périphériques embarqués. Notre choix de se concentrer sur ces composants est issu du fait que ces derniers représentent la partie principale et critique à prendre en charge en terme de consommation énergétique. En effet, les composants périphériques représentent la seule façon pour un objet connecté d'interagir avec son environnement de déploiement. Ils sont ainsi constamment utilisés et sollicités par l'application embarquée sur l'objet connecté.

Ainsi, l'objet utilisera souvent différents capteurs (e.g. température, gaz, etc.) pour recueillir diverses informations sur son environnement de déploiement. Ces données sont traitées localement et/ou envoyer à des stations distantes en utilisant un moyen de communication qui est le plus souvent sans-fil (e.g. WiFi, cellulaire, LoRa, SigFox, etc.). Par ailleurs, de par des commandes reçues de l'extérieur ou de sa propre initiative, l'objet peut être amené à activer des actionneurs (e.g. moteur, ventilateur, etc.). À cause de ce schéma d'utilisation intensif des composants périphériques, nous les identifions comme un acteur principal et majeur dans la consommation énergétique globale d'un objet connecté. Cette affirmation devient d'autant plus vrai lorsqu'on se focalise sur les petits objets connectés d'aujourd'hui. Ceux-ci sont souvent composés de microcontrôleurs basse consommation qui sont utilisés pour effectuer des tâches basiques et bien précises en comparaison avec les processeurs grand public. Malheureusement, concernant les composants périphériques embarqués, malgré beaucoup d'avancées sur la miniaturisation et la consommation énergétique de ceux-ci, la dissipation énergétique engendrée reste tout de même conséquente. En effet certains éléments architecturaux ne peuvent être facilement optimisés. Ainsi, par exemple, un module de communication voulant transmettre à très longue distance ou à très haute sera souvent dans l'obligation d'utiliser une quantité d'énergie significative.

Notre objectif est donc d'aider le développeur à comprendre et appréhender le comportement énergétique de ces composants périphériques afin d'être le plus efficace et économe possible dans leur utilisation en terme énergétique. À cet effet, nous pensons fortement qu'un processus d'inférence de modèles énergétiques pour les composants périphériques peut venir en support du développeur embarqué dans cette compréhension. Généralement, dans la littérature, la création de modèles énergétique est une tâche longue et difficile. La phase de création nécessite une longue période d'acquisition de divers échantillons énergétique. La deuxième phase représente la validation du modèle énergétique en prouvant son adéquation dans des conditions environnementales réelles. Pour surmonter cette barrière et simplifier la manipulation des modèles énergétique générés, nous décidons d'aborder un niveau d'abstraction plus élevé et de faire en sorte que les modèles énergétique de chaque composant périphérique soit centré sur l'interface de programmation applicative (API) de chaque pilote de périphérique.

Le choix de se focaliser sur les composant périphériques et de centrer les modèles énergétiques sur l'API du pilote logiciel de ces derniers est motivé par plusieurs aspects résumés ci-dessous :

— Les composants périphériques embarqués (i.e. les opérations d'entrées/sorties) représente la majeure partie de la consommation énergétique d'un objet connecté. Par conséquent, pour aider le développeur, nous devons nous concentrer sur la consommation énergétique de ces périphériques.

— Les pilotes logiciels des composants périphériques embarqués sont l'interface principal entre l'application embarquée et le matériel. Ils sont ainsi responsables de toutes les activités du composant périphériques. De ce fait, ils suffisent souvent à résumer le comportement fonctionnel d'un composant.

— Lors du développement d'une application embarquée, les développeurs embarqués sont directement confrontés à l'API des pilotes logiciels afin d'exploiter les composants périphériques. L'obtention de modèles énergétiques centrés sur cet API semble donc être l'aspect le plus naturel à proposer au développeur.

— À travers la prise en considération du contexte industriel actuel du secteur des objets connecté, nous remarquons une part significative de composants périphériques livrés avec des sources fermées. Ce fait ne permet pas donc au développeur de comprendre les optimisations et le fonctionnement du composant via la lecture du code source logiciel. A cet effet, les modèles énergétique proposés peuvent donc être utilisés avec l'interface logicielle fournie avec le composant afin d'en explorer et en apprendre plus sur son comportement.

— Dans une majorité des cas, ces pilotes logiciels embarqués sont léger et contiennent un nombre limité de fonction logicielles. En effet, de par la nature des petits objets connecté et la course à la diminution des coûts, les composants périphériques utilisés sont souvent bornés en terme de fonctionnalités. Cet aspect nous permet d'assurer une légèreté dans l'utilisation de notre processus d'inférence de modèles énergétique.

— L'aspect simple et naturel des modèles énergétiques générés induit un temps de génération raisonnable. Ainsi, des réévaluations et des régénérations rapides sont possibles concernant ces modèles. Cela permet de prendre en compte plus aisément de nouveaux attributs de développement. Nous pensons notamment

à un changement d'environnement et de conditions de déploiement (e.g. bruit radio de l'environnement à prendre en compte), ou encore la modification de la plate-forme embarquée cible (i.e. ajout d'un nouveau composant périphérique).

Nos motivations vont encore plus loin. En effet, les modèles énergétique résultants peuvent être utilisés pour fournir au développeur embarqué un ensemble de documentation énergétique pour chaque périphérique embarqué. La documentation fournie peut grandement différer d'une fiche technique (i.e. ang. *datasheet*) du fabricant car la première est basée sur des mesures effective de la consommation énergétique lors de l'exécution de l'application et prend en compte les conditions de l'environnement de déploiement (bruit radio, interaction avec la plate-forme cible, etc.). En outre, une fiche de technique d'un composant n'est pas toujours conforme aux chiffres de consommation énergétique réelle [LMGF13].

6.3 Proposition

Comme précédemment énoncé, nous proposons comme solution à la problématique de compréhension énergétique du développeur, un processus d'inférence de modèles énergétique pour composants périphériques embarqués. Nous pensons et nous affirmons que ce processus peut venir en aide et en support au développeur pour gagner une meilleur compréhension du matériel utilisé et lui permettre de livrer un logiciel efficace en énergie.

En revenant sur le cycle de développement précédemment présenté et illustré une nouvelle fois ci-dessous (cf. Figure 6.1), nous plaçons notre processus d'inférence dans la partie supérieur de ce dernier. En effet, nous tirons avantage de notre framework de cartographie énergétique présenté dans le chapitre III afin de réaliser le processus d'inférence de modèles présentés dans ce chapitre. Plus précisément, nous utilisons le framework de cartographie afin de réaliser plusieurs passes de mesure énergétique. Cela nous permet de collecter plusieurs échantillons de consommation énergétique du composant périphérique cible où chaque échantillon décrit une ou plusieurs fonctionnalités du composant. Le framework de cartographie effectue le profilage énergétique du système cible et du composant périphérique en condition réelle d'exécution. Le principal avantage est donc de pouvoir enregistrer le comportement énergétique réel de chaque composant en prenant ainsi en compte l'impact de l'environnement de déploiement (i.e. bruit radio, bruit généré par le circuit, etc.). Les modèles énergétique générés après le traitement des échantillons collectés tendent à résumer le comportement énergétique de chaque périphérique. Notamment, les états énergétiques existants et les transitions entre chaque état.

Le processus d'inférence de modèles énergétiques est illustré globalement dans la Figure 6.2. Ce processus passe par plusieurs étapes. Nous décrivons et nous détaillons chacune de ces étapes plus loin dans ce chapitre. Globalement, nous pouvons résumer le fonctionnement global de notre processus en donnant un schéma d'utilisation classique. Ainsi, pour un composant périphérique donné, le développeur doit sélectionner le pilote logiciel de ce dernier, puis annoter toutes les fonctions logicielle qu'il veut inclure dans le modèle énergétique. Une application tierce conçue grâce au framework de compilation Clang analyse les annotations et génère plusieurs fichiers de code source de test (.ang *benchmark*). Chaque fichier représente en fait un fichier de test composé d'une liste de fonctions provenant du pilote logiciel du composant de par les annotations du développeur. À ce stade, le framework de cartographie énergétique décrit dans la section 3 est utilisé pour instrumenter chaque

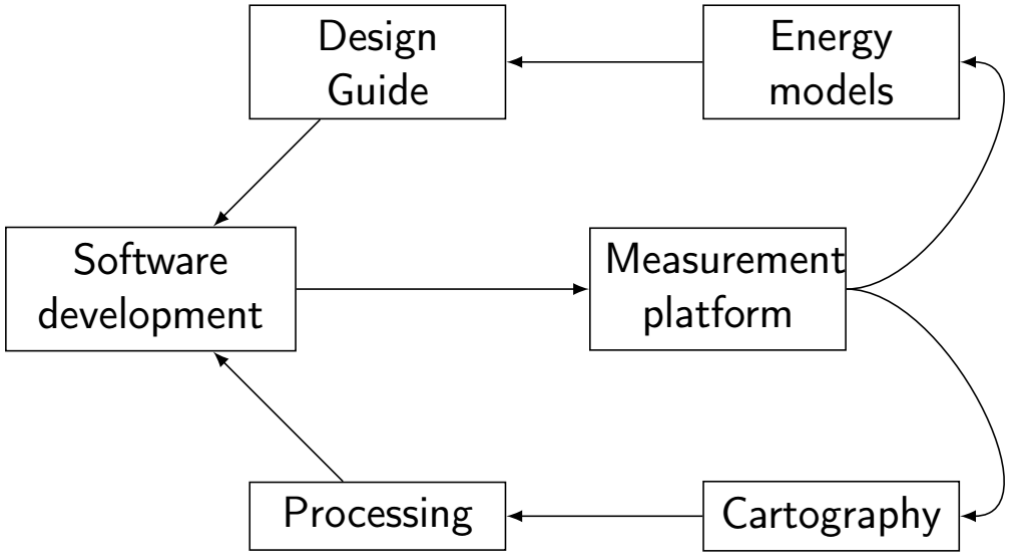


FIGURE 6.1 – Le cycle de développement centré sur la ressource énergétique.

fichier source de test, puis évaluer la consommation énergétique résultante lors de l'exécution sur le système embarqué cible. Les mesures énergétique peuvent être effectuées à plusieurs reprises (i.e. paramètre développeur) puis moyennées afin d'avoir statistiquement une meilleurs résistance au bruit de mesure.

Les traces énergétiques résultantes de la phase de mesure et profilage sont ensuite traitées. Dans un premier temps, le but est de détecter tous les états énergétiques qui sont contenus dans les traces énergétiques. Pour cela un algorithme de détection d'état est utilisé. Ensuite, une étape de regroupement d'états similaires est effectuée afin de réduire le nombre d'états existants. Enfin, grâce au profilage effectué via notre framework de cartographie, un modèle énergétique basé sur des automates à état est généré, en faisant la lien entre les états énergétique existant et les fonctions du pilote logiciel déclenchant les transitions entres ces états.

6.4 Auto-génération des programmes de test et de stress du périphérique embarqué

Nous présentons ici la première étape de notre processus d'inférence de modèles. Celle-ci représente la génération des fichier de code source de test (ang. *benchmark*). L'objectif de cette étape est de simplifier pour le développeur la génération des échantillons de test du composant périphérique embarqué à étudier. Les fichier de test générés de par cette étape seront utilisés comme base d'échantillons dans la prochaine étape de mesure énergétique.

6.4.1 Annotation des fonctions

Dans le but d'effectuer la génération des fichiers de test, nous fournissons au développeur embarqué, afin de le guider, un outil sous forme de langage d'annotation basé sur le format JSON. Ce langage se veut très léger et

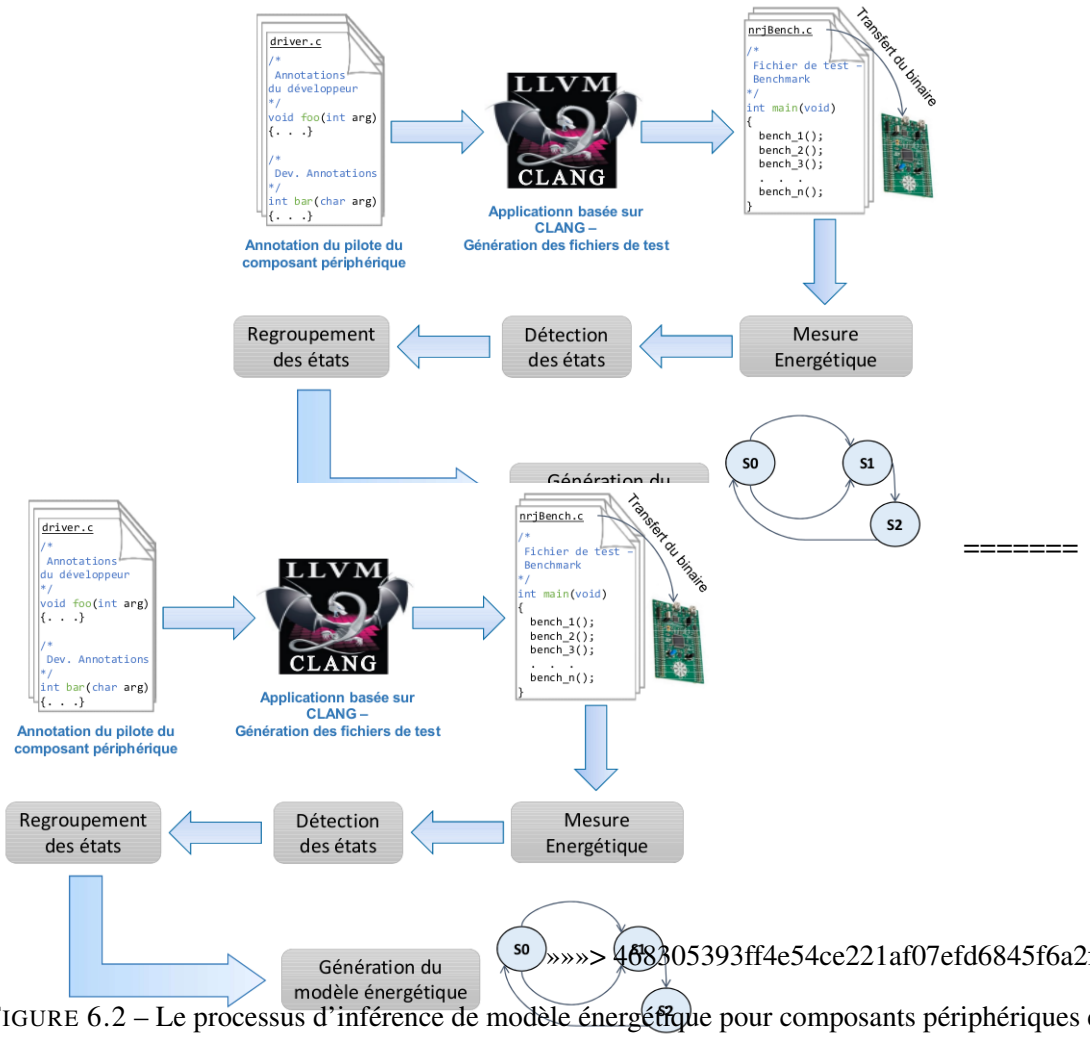


FIGURE 6.2 – Le processus d'inférence de modèle énergétique pour composants périphériques embarqués.

simple d'utilisation par le développeur dans sa tâche de génération des tests. Pour chaque composant périphérique embarqué ciblé par le développeur, le générateur de tests prend en entrée les sources du pilote du périphérique en question contenant les fonctions annotées par le développeur. Le générateur analyse les annotations et génère en sortie plusieurs fichiers de test. Ces derniers sont à leur tour instrumentés dans l'étape suivante, en utilisant le processus de profilage de notre framework de cartographie présenté dans le chapitre 3. Plus précisément, chaque échantillon de test est énergétiquement profilé afin d'en extraire les états énergétiques qui formeront les modèles.

Concernant le mini-langage d'annotation proposé au développeur embarqué. Celui-ci lui permet de guider la génération des fichiers de test. Le développeur peut sélectionner le type de la fonction annotée ainsi que l'espace des arguments à explorer pour chaque paramètres de la fonction. Optionnellement, le développeur peut aussi préciser des fonctions nécessaires à appeler avant ou après la fonction annotée, en précisant leurs arguments. Le type de fonction explicité dans l'annotation n'a aucun lien avec le typage au sens langage de programmation. Il sert à diriger et à guider la génération des fichiers de test et plus précisément, le placement des fonctions

annotées dans chaque fichier généré. Ce schéma de génération est expliqué plus tard dans cette section. Le développeur peut choisir le type de la fonction annotée parmi 4 possibilités : INIT, DEINIT, CONF, ACTIVE. Nous en donnons la description ci-dessous :

— Les fonctions de type INIT (i.e. Initialisation) sont simplement celles qui sont responsables de la mise sous tension et/ou initialisation du composant périphérique ciblé.

— À l'opposé, celles de type DEINIT (i.e. Dé-initialisation) pilotent la mise hors tension et/ou la désactivation du composant.

— Les fonctions de type ACTIVE (i.e. Activité) représentent une charge de travail effective de la part du composant périphérique. Cette charge ne modifie pas l'état interne du composant périphérique embarqué. Nous pouvons citer, comme exemple, l'envoi ou la réception de trames de communication par un module de communication.

— Les fonctions de type CONF (i.e. Configuration) sont les fonctions ayant la possibilité de modifier l'état interne du composant périphérique. Autrement dit, les fonctions de type ACTIVE exécutées dans un état interne différent peuvent exposer un comportement énergétique différent. Nous pouvons citer par exemple, la modification de la puissance d'émission ou du canal fréquentiel d'un composant sans fil de communication.

Le format JSON de l'annotation pouvant être faite par le développeur est exposé ci-dessous :

```
/** !MODELGEN! {
 "id":        "un_ID",    # pour l'identification des composants
 "funcType":  "ACTIVE",   # ou CONF, INIT, DEINIT
 "prevFuncs": [],         # liste des fonctions a appeler avant, avec leurs arguments
 "nextFuncs": [],         # liste des fonctions a appeler apres, avec leurs arguments
 "args": {                # description de l'espace des arguments
  "1": {"value": val1},
  "2": {"value":{"inf":0, "max":2000, "step":100}}
}} */
void func(type_1 arg1, type_2 arg2);
```

6.4.2 Schéma de génération des fichiers de test

L'étape venant d'être décrite permet au développeur embarqué de spécifier les fonctions qu'il veut inclure dans le modèle énergétique final ainsi que l'espace des arguments de fonction à explorer. Néanmoins, cela n'est pas suffisant pour générer les fichier de tests nécessaires à la génération des modèles. Dans cette étape, le typage des fonctions dans les annotations faites par le développeur prend une place importante. En effet, la génération des fichiers de tests nécessite aussi de savoir comment organiser les fonctions annotées par le développeurs à l'intérieur de ces même fichiers. En ayant à l'esprit la définition faite de chaque type de fonction dans les annotations, nous illustrons en Figure 6.3 le processus suivi afin de générer chaque fichier de test (ang. *benchmark*).

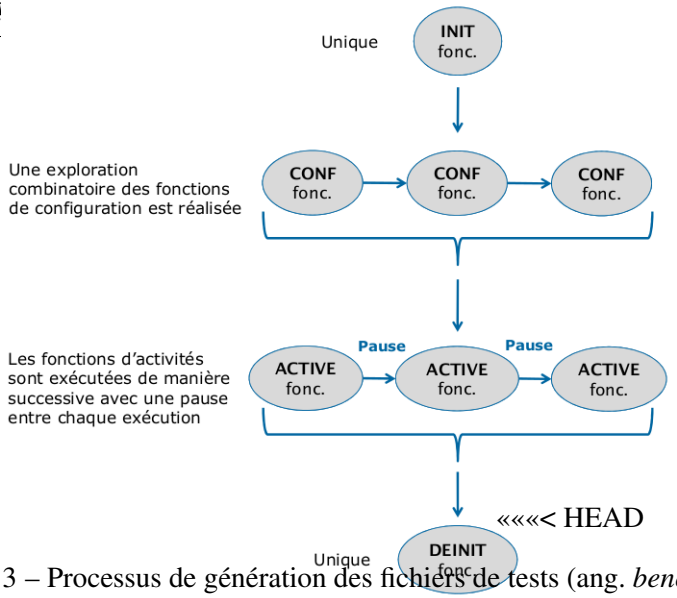


FIGURE 6.3 – Processus de génération des fichiers de tests (ang. *benchmark*)

<<<<<< HEAD

=======

FIGURE 6.4 – Processus de génération des fichiers de tests (ang. *benchmark*)

>>>>> 468305393ff4e54ce221af07efd6845f6a2f2d09

La génération suit ainsi un chemin spécifique afin de créer tous les fichier de test. En résumé, chaque fichier de test est composé d'une unique fonction de type "INIT (i.e. initialisation)", puis plusieurs fonctions de type "CONF (i.e. configuration)", plusieurs fonctions de type "ACTIVE (i.e. activité)" et enfin, une unique fonction de type "DEINIT (i.e. dé-initialisation)" Plus précisément, le processus de génération liste toutes les combinaisons (i.e. au sens mathématique) possibles des fonctions de configuration en incluant tout l'ensemble des valeurs des arguments précisé lors de l'annotation par le développeur. Pour chacune des combinaisons, un nouveau fichier de test est généré par le processus qui y inclut la combinaison en question. Pour le reste, tous les fichiers de test partagent et contiennent les mêmes fonctions d'initialisations, d'activités et de dé-initialisations. Concernant les fonctions d'activités, elle sont intégrées à tous les fichiers de test de la même sorte. Ces fonctions sont ainsi appelées dans chaque fichier de manière successive avec une boucle sur l'espace des arguments décrit par le développeur. Les exécutions successives de ces fonctions d'activités sont espacé par un temps de pause définit par le développeur.

Le schéma suivi pour la génération des fichiers de test peut être directement expliqué de par la définition de chaque type de fonctions dans les annotations. En effet, les fonctions de configuration sont définis comme des fonctions permettant le changement de configuration d'un composant périphérique embarqué. De par ce changement de configuration, un changement dans l'état interne du périphérique se produit. Un composant périphérique se trouvant dans un état interne différent peut avoir un comportement énergétique différent lors de de la réalisation de taches actives (i.e. envoi/réception de donnée, écoute de l'environnement, etc.). Ainsi, pour ne pas manquer un état interne d'un composant périphérique, un listing combinatoire des fonctions de configuration doit être fait comme décrit dans le schéma de génération ci-dessus. Quant aux fonctions d'activés, elle sont définies comme les déclencheurs des tâches actives du périphériques. Néanmoins, celles-ci ne sont pas censées modifier l'état interne du périphérique. Ainsi, une fonction d'activité exécutée plusieurs fois dans le

même état interne du périphérique observe à chaque fois le même comportement énergétique (ssi. les conditions de déploiement sont les mêmes). Pour cette raisons, ces fonctions peuvent juste être exécutées successivement.

Le schéma combinatoire du processus de génération de fichiers de test peut être considéré comme très long en terme de durée. Néanmoins, ce schéma est aussi mit en place en partant de l'observation des petits systèmes embarqués sur lesquels notre travail se porte. Dans ce domaine d'étude, les observations faites à travers nos expérimentations nous permettent de poser l'hypothèse, que la couche logiciel du pilote d'un périphérique embarqué est souvent légère. Autrement dit, elle est composée d'un nombre limité de fonctions, permettant ainsi à cette couche logiciel d'être embarqué dans de très petits objets contraints en puissance et mémoire. En conséquence de cette légèreté, malgré l'aspect combinatoire du schéma de génération, le nombre de fichiers de test produits reste raisonnable.

6.5 Sélection et regroupement des états énergétiques

Nous décrivons dans cette section, notre processus de détection des états énergétiques dans le comportement de chaque composant périphérique. Cette étape représente le cœur de notre approche d'inférence de modèles énergétiques. L'objectif est d'isoler et de déterminer la consommation énergétique et les différents niveaux d'énergie induits par l'exécution de fonctions particulières du pilote logiciel qui contrôle le composant périphérique embarqué.

6.5.1 Mesure et profilage énergétique

Les fichiers de test issus de l'étape précédente sont embarqués et exécutés sur la plate-forme embarquée cible. Chaque fichier, une fois sur la plate-forme cible, donne lieu à l'exécution d'une application de test ayant comme but le stress du composant périphérique. Lors de cette exécution, la consommation énergétique du système embarqué cible est mesurée et profilée via le framework de cartographie énergétique. Ainsi, En résultat à la phase de mesure énergétique des tests, nous obtenons plusieurs traces énergétiques représentant le comportement énergétique de chaque application de test profilée.

Les traces énergétiques citées sont vues et traitées comme des séries temporelles (ang. *time series*). La définition de celles-ci a été faites dans le chapitre 3. Nous complétons la définition d'une trace énergétique avec celle d'un état énergétique. Nous définissons un **état énergétique** comme une sous-partie contiguë d'une trace énergétique qui dénote une coupure effective dans le comportement en comparaison avec le reste de la trace.

6.5.2 Détection des états énergétiques

Nous décrivons ici de manière effective la méthode de détection des états énergétiques utilisé. Dans le chapitre 3, nous avions discuté d'un sujet connexe. Plus précisément, nous avons étudié les variations dans le comportement énergétique (i.e. courbe de courant) de chaque fonction logicielle embarqué instrumentée. En effet, l'objectif était de voir s'il y avait beaucoup de variations, de pics, de changements soudains et donc une grande variabilité dans le comportement énergétique d'une fonction. En acquérant cette connaissance, nous avons pu affirmer de l'utilité de l'exploration ou non des sous-fonctions appelées par cette fonction. Pour atteindre cet

objectif, la méthode utilisée était simple. Ainsi, l'utilisation d'une approche basée sur la dérivé (i.e. en terme mathématique) de la courbe de courant de la fonction est suffisante. Néanmoins l'objectif n'était pas critique.

Dans l'optique d'une détection des états énergétiques pour la construction de modèles énergétique, les choses sont différentes. En effet, nous voulons détecter les états tout en gardant, en même temps, une bonne résistance contre le bruit et les pics pouvant être présents dans la trace énergétique produite après la phase de mesure. Le but n'est donc plus d'étudier la variabilité présente dans la courbe énergétique d'une fonction logicielle. l'objectif est de sélectionner des sous-parties contiguë d'une trace énergétique dans laquelle les caractéristiques statistiques restent identiques. Lors du changement de ces caractéristiques, une coupure est dénotée délimitant ainsi un état énergétique.

6.5.3 Définition et état de l'art des approches

Dans le cadre de notre étude, nous avons choisi de nous reporter sur le domaine des méthodes de détection de point de changement (ang. *changepoint*). Ces approches sont pour le plus souvent utilisées dans le domaine financier pour l'étude de la volatilité des marchés financiers [AIL99][AG02][Fer04], en bio-informatique dans l'étude du génome humain [BBM00][OVLW04][PRL+05], en climatologie [NAJ12] ou encore en océanographie [KAEEJ10]. Un example simple de point de changement est illustré sur la figure 6.5. Ces domaines sont certes très loin de notre cas d'étude dans cette thèse. Néanmoins, quelque soit le domaine d'application, ces approches s'appliquent toujours sur des séries temporelles. Partant de ce point, il n'y a donc pas de contradictions ou d'inconvénients à utiliser et appliquer ces approches sur des traces énergétiques de mesure. Ce cas d'application n'a, à notre connaissance, jamais était réalisé ou utilisé dans l'état de l'art.

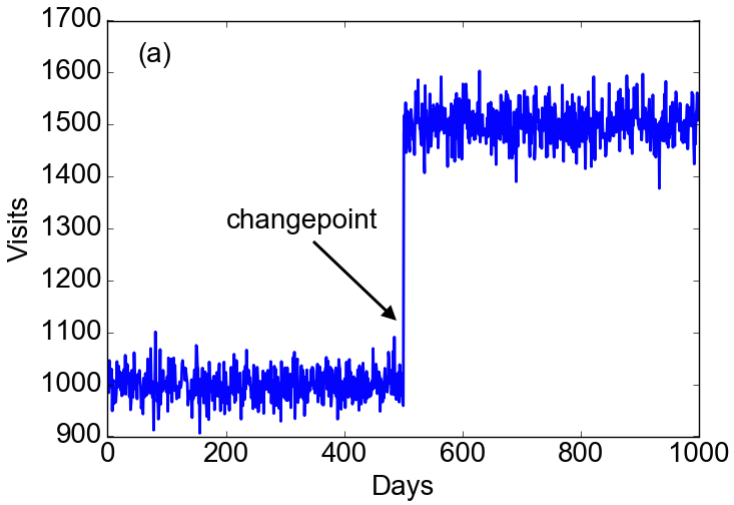


FIGURE 6.5 – Une illustration d'un point de changement sur un cas simple d'étude du nombre de visites d'un commerce.

Nous donnons dans ce qui suit la définition d'un *point de changement* qui complète ainsi la définition donné pour un état énergétique dans une trace énergétique :

Definition. (Point de changement) Soit une série de données ordonnées, représentant dans notre cas une trace énergétique (i.e. intensité du courant en fonction du temps), $y_{1:n} = (y_1, ..., y_n)$. Un point de changement se produit dans cette série s'il existe un point de temps, $t \in 1, ..., n-1$, tel que les propriétés statistiques de $(y_1, ..., y_t)$ et de $(y_{t+1}, ..., y_n)$ sont différentes.

Plusieurs approches ont été proposées afin de résoudre la détection d'un **unique** point de changement. Nous pouvons référer à [Hin70] et [JG87] comme les premières études en ce sens. Celles-ci utilisent un test statistique basé sur un rapport de vraisemblance (ang. *likelihood ratio*).

Néanmoins, une trace énergétique pouvant évidement contenir plusieurs états et de ce fait plusieurs points de changement, nous étendons la définition précédente à de multiple points de changement :

Definition. (Multiple points de changement) Soit une série de données ordonnées, $y_{1:n} = (y_1, ..., y_n)$. Soit m le nombre de points de changement existants dans cette série, tel que leur position est $t_{1:m} = (t1, ..., t_m)$, avec $[1:m] \in [1:n-1]$. Pour $i, j \in [1:m]$, $t_i < t_j$ si et seulement si $i < j$, les m points de changement divisent la série de données en $m+1$ segments avec le i-ème segment contenant $y_{(t_{i-1}+1):t_i)}$. Chaque segment se voit caractérisé par un ensemble de paramètres θ les différenciant des autres segments et précisant ainsi le point de changement.

L'approche la plus souvent utilisée afin d'identifier de multiple points de changement est la minimisation d'une fonction de coût. La recherche et l'identification de la meilleur segmentation possible d'une trace énergétique (i.e. série temporelle) est donnée par 6.1 :

$$Q_\beta(y_{1:n}) = min_t \sum_{i=1}^{m+1} [C(y_{(t_{i-1}+1):t_i)} + \beta f(m)] \tag{6.1}$$

C représente ici la fonction de coût pour chaque segment $y_{(t_{i-1}+1):t_i)}$ et $\beta f(m)$ constitue la pénalité associée à la fonction de coût afin de limiter la sur-détection de points de changement. Nous dériverons plus précisément de cette pénalité dans la prochaine sous-section. Concernant la fonction de coût, le critère le plus utilisé (i.e. aussi utilisé dans notre processus) reste aussi l'estimateur du maximum de vraisemblance, ou plus précieusement le logarithme de la vraisemblance qui est plus simple à manipuler lors des calculs 6.3. La fonction de coût à minimiser devient ainsi **??**

$$l(y_{(t_{i-1}+1):t_i)}) = max_\theta logp(y_{(t_{i-1}+1):t_i)}; \theta) \tag{6.2}$$

$$C(y_{(t_{i-1}+1):t_i)}) = max_\theta logp(y_{(t_{i-1}+1):t_i)}; \theta) \tag{6.3}$$

Typiquement, Le but est d'analyser deux segments consécutifs de données appartenant à la série temporelle. Puis, définir si le point de liaison entre ces segments représente bien un *point de changement*. Cela est le cas si les deux distributions (i.e. les deux segments de données) différent "assez". Une hypothèse est posée sur le fait que ce point est un point de changement ou non. La fonction de coût évalue alors cette hypothèse via le rapport de vraisemblance de la distribution afin de mesurer l'adéquation de cette dernière à l'hypothèse émise. De ce fait, décider si oui ou non il y a un point de changement.

Dans l'état de l'art des méthodes de détection de points de changement, il en existe deux qui sont fréquemment citées et utilisées. La première nommée *Binary Segmentation (BS)* proposé par Scott et al. [SK74]. Celle-ci

représente plus une extension à la recherche d'un unique point de changement en effectuant l'opération de manière itérative à des sous-ensembles des données en entrée. BS est initialement appliquée à l'ensemble des données (i.e. la série temporelle). Si un point de changement est identifié, l'ensemble est divisé en deux segments décrivant les données après et avant le point de changement. BS est ainsi répété et appliqué de nouveau à chaque segment jusqu'à ce que qu'aucun point de changement ne soit trouvé. Autrement dit, BS essaye de minimiser l'équation de coût 6.1 à chaque étape en essayant ainsi d'ajouter un nouveau point de changement. L'avantage est l'efficacité du calcul avec une complexité en O(nlog(n)). Cela contraste avec sa précision quant aux points de changement détectés. En effet, la méthode de recherche de BS ne permet pas de garantir l'identification d'un minimum global pour le problème d'optimisation en 6.1, mettant ainsi en doute la précision de la détection.

Dans l'objectif d'une meilleure précision, l'approche *Segment Neighbourhood (SN)* est proposé dans [AL89] comme étant une méthode exacte à l'inverse de l'approche BS. Ainsi, La méthode SN recherche dans tout l'ensemble des segments possibles en utilisant la programmation dynamique. Celle-ci définit une limite supérieur à l'espace des segments (i.e. le nombre maximum de points de changement possible). Puis, elle calcule la fonction de coût pour tout les segments possibles. La recherche exhaustive de tout l'espace de segment assure l'exactitude de la méthode. Malheureusement, une conséquence en est le coût significatif de cette recherche qui est de complexité O(n^2). De plus dans le pire des cas, la méthode SN atteint une complexité de calcul en O(n^3).

De par l'étude de ces méthodes nous pouvons arriver à un constat simple. Les méthodes décrites sont soit *rapide mais donnant un résultat approximatif*, ou alors *lente avec un résultat exact*. La question qui se pose est donc : Comment identifier tout les points de de changement dans l'espace des segments possibles ($2^n - 1$, avec n la taille de la trace énergétique), tout en gardant une complexité de calcul raisonnable ? Cette question a été grandement résolue par la méthode PELT que nous présentons dans ce qui suit.

6.5.4 L'approche PELT

Pour atteindre l'objectif fixé et répondre à la question posé précédemment, nous avons choisi d'utiliser un algorithme de détection multiple de points de changement proposé par Killick et Haynes dans [KFE12]. Celui-ci est nommé PELT pour *Pruned Exact Linear Time*. En effet cette méthode permet de grouper les avantages des méthodes SN et BN sans leurs principales inconvénients. Ainsi, les principaux avantages de cette méthode sont l'exactitude des résultats et la vitesse de recherche dans l'espace des segments possibles (i.e. de taille $2^n - 1$ avec n la taille de la trace énergétique). La méthode PELT est une méthode paramétrique dite "*exacte*". Á l'instar des autres méthodes citées précédemment, l'approche PELT effectue aussi la détection des points de changement en minimisant une fonction de coût 6.1. L'estimateur du maximum de vraisemblance d'un possible segment de la trace énergétique est pris comme étant le coût à minimiser.

La méthode PELT se base une grande partie sur une précédente approche proposée par Jackson et al. nommée *Optimal Partitioning (OP)* **??**. Celle-ci repose sur une approche de programmation dynamique afin de résoudre la minimisation 6.1. L'idée de base de cette méthode peut se résumer comme suit :

— Ajouter les points de données de la série temporelle, un point à chaque étape, puis calculer la segmentation/partition optimale pour $y_{1:t}$ pour $t = 2, 3, ..., n$.

— Utiliser le fait de connaître l'existence d'un point de changement en τ, permet de segmenter indépendamment les données de la série avant et après τ.

— Afin de calculer la partition optimale pour $y_{1:t}$, On utilise la connaissance des partitions précédentes.

Plus précisément, la méthode OP commence par un conditionnement sur le dernier point de changement. Elle relie ensuite la valeur optimale de la fonction de coût au coût de la partition optimale des données avant le dernier point de changement, plus le coût pour le segment du dernier point de changement à la fin des données. De manière plus formelle :

Definition. (Optimal Partitioning (OP))
On définit :
$P_t = \tau : 0 < t_1 < ... < t_m < t$
on pose :
$F(t) = min_{\tau \in P_t} \sum_{i=1}^{m+1} [C(y_{(\tau_{i-1}+1):\tau_i}) + \beta], avec f(m) = m$
Cela peut s'écrire donc :
$F(t) = min_{\tau *} min_{\tau \in P_{\tau *}} \sum_{i=1}^{m} [C(y_{(\tau_{i-1}+1):\tau_i}) + C(y_{(\tau *+1):t}) + \beta]$
La nouvelle fonction à minimiser devient donc :
$F(t) = min_{\tau *} F(\tau *) + C(y_{(\tau *+1):t}) + \beta$

De cette définition, nous pouvons remarquer que cette nouvelle écriture dénote une récursion. Cette récursion peut être résolue pour $t = 1, ..., n$ de manière quadratique (i.e. $O(n^2)$) par rapport à n (taille/longueur de la série temporelle et donc de notre trace énergétique).

La méthode PELT que nous utilisons pour notre processus se base sur la méthode OP et part du même principe concernant la récursion à calculer. Elle y ajoute une simple étape de réduction afin d'en accélérer la vitesse de traitement et d'en réduire donc la complexité du calcul. L'idée principale de la méthode PELT est : si on a connaissance d'un point de changement "évident" d'une série temporelle à un certain point de temps noté s, cela signifie que pour chaque $T > s$, le point de changement le plus récent ne peut être à $t < s$. Cette simple idée décrit l'étape de réduction de la méthode PELT, permettant d'omettre la recherche de l'espace $t < s$.

Nous illustrons graphiquement cette étape de réduction en Figure 6.6. Cette illustration dénote un point de changement "évident" en s. Pour chaque $T > s$, la meilleur partition qui implique aussi un point de changement en s sera plus grande que celle ayant $[t, T]$ comme unique segment. Ainsi t ne peux jamais être le point de changement optimal le plus récent avant T pour tout $T > s$. Ainsi, la méthode PELT est d'abord appliquée à toute la série temporelle, puis itérativement et indépendamment à chaque partition, jusqu'à ce qu'aucun autre point de changement ne soit détecté.

De ce fait, si plusieurs t sont réduit et donc exclue de la fonction de minimisation, le temps de calcul se retrouve grandement réduit et la complexité devient alors en $O(n)$. Afin d'atteindre cet état, il est uniquement primordial que le nombre de point de changement augmente linéairement avec l'accroissement de n dans la série temporelle. Cela est totalement correspondant à notre cas d'étude, où les traces énergétiques contiennent différents comportements au grès des fonctions exécutées.

Nous pouvons donc affirmer que l'approche PELT est plus précise que les méthodes de recherche approximatives (e.g. **BS**) et plus rapide par rapport aux autres méthodes de recherche exactes citées précédemment (e.g. **SN** ou **OP**).

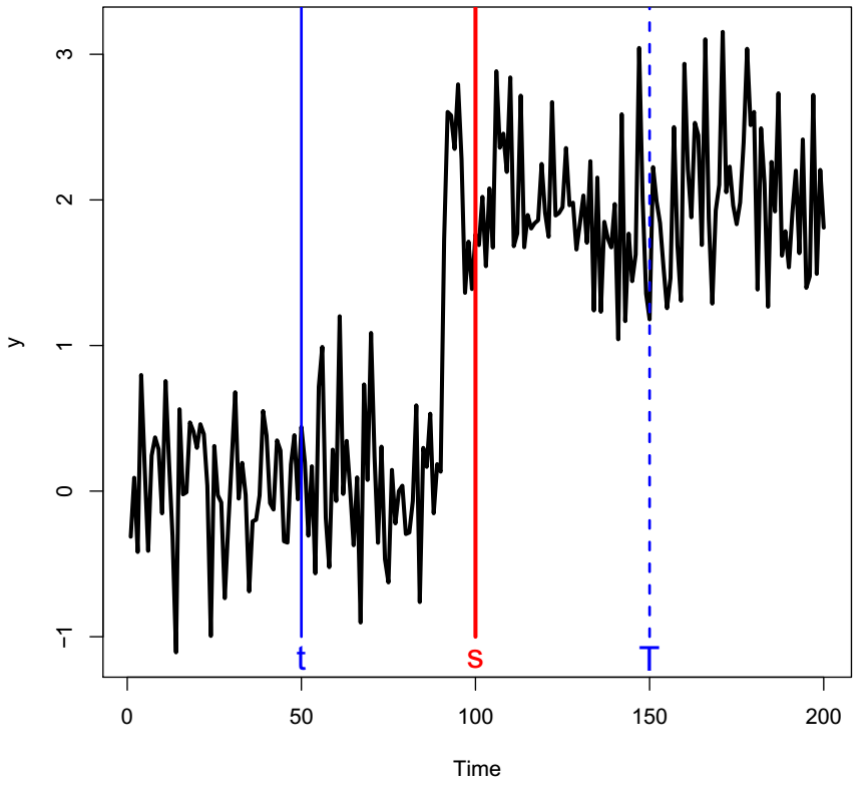


FIGURE 6.6 – Étape de réduction de la méthode PELT.

Dans notre processus d'inférence, chaque point de changement identifié par l'approche PELT indique la fin d'un état énergétique effectif ainsi que le début d'un nouveau sur la trace énergétique. Enfin, afin d'accélérer le processus de détection d'état ainsi que sa fiabilité nous appliquons en prétraitement à la trace énergétique un échantillonnage aléatoire. Techniquement, Cela équivaut à sélectionner aléatoirement, à intervalle régulier sur la trace, un point. Typiquement, pour une trace obtenu après mesure énergétique matérielle via un échantillonnage de 50 kHz, nous effectuons un second échantillonnage (logiciel) aléatoire de 1/10 (i.e. Sélection d'un échantillon par intervalle de 10 échantillons). De façon empirique, nous avons déduit que ce prétraitement est dans tout les cas positif. Il permet une élimination des pics résiduels sur la trace facilitant la détection des états énergétiques. Les points éliminés sont de nouveau pris en compte dans la trace énergétique après la phase de détection d'état afin de ne pas fausser le calcul des métriques énergétiques.

6.5.5 Pénalité de la détection de points de changement

L'approche PELT, à l'instar des autres des approches de détection de point de changement, se base sur la minimisation d'une fonction de coût afin d'achever son objectif de recherche. Néanmoins, comme défini 6.1, en plus de cette fonction de coût, l'approche PELT prend en entrée un terme spécifique ($\beta f(m)$) qui fait référence à une pénalité (ang. *penalty*). Celui-ci joue un rôle important dans la détection des points de changement. Le terme de pénalité utilisé afin d'éviter la sous/sur segmentation . Autrement dit, ce terme aide à contrôler le nombre

de points de changement détectés. Il existe de nombreuses méthodes automatiques (ou semi-automatiques) pouvant être utilisées afin de fixer la valeur et comportement de cette pénalité (entre autre : Akaike Information Criterion-AIC", Schwarz Information Criterion-"SIC", Bayesian information criterion-"BIC", Modified Bayes Information Criterion-"MBIC", "Hannan-Quinn" ou "Asymptotic) [**?**]. Néanmoins cela ne correspondant pas à l'ensemble et l'hétérogénéité des traces énergétiques existantes. En effet, Le choix d'une pénalité appropriée reste encore à ce jour une question ouverte et dépend de nombreux facteurs, dont le nombre de points de changements et la longueur des segments. Les deux sont inconnus avant l'analyse.

Pour fixer la valeur de ce terme de pénalité, nous avons décidé d'utiliser une approche dite manuelle. L'utilisation d'une pénalité manuelle s'accompagne d'une règle de base : une valeur de pénalité basse entraîne l'identification d'un plus grand nombre de points de changement, tandis qu'une pénalité plus élevée entraîne, au contraire, une identification de moins de points de changement. Pour définir la valeur de la pénalité la plus appropriée pour une trace d'énergie donnée, nous suivons une méthode assez simple dite en "arc" (ang. *elbow method*). Cette méthode est définie par la variation de la valeur de la pénalité ainsi que la variation du nombre de points de changement identifiés par rapport à la pénalité utilisée.

Comme nous pouvons le voir dans la figure 6.7, le nombre de points de changement détectés subit à une diminution rapide qui représente les points de changement qui sont principalement induits par le bruit et ralentira de plus en plus jusqu'à atteindre un nombre beaucoup plus bas de points de changement identifiés. Concernant la valeur de pénalité la plus appropriée à choisir. Nous identifions la valeur de pénalité du *"coude de l'arc"* comme la valeur donnant les résultat les plus satisfaisants par rapport à la reconnaissance des segments effectif dans la trace énergétique. Afin de trouver ce point automatiquement, une méthode simple est de calculer la dérivée seconde de la courbe en arc afin et de prendre le point d'abscisse de valeur maximum.

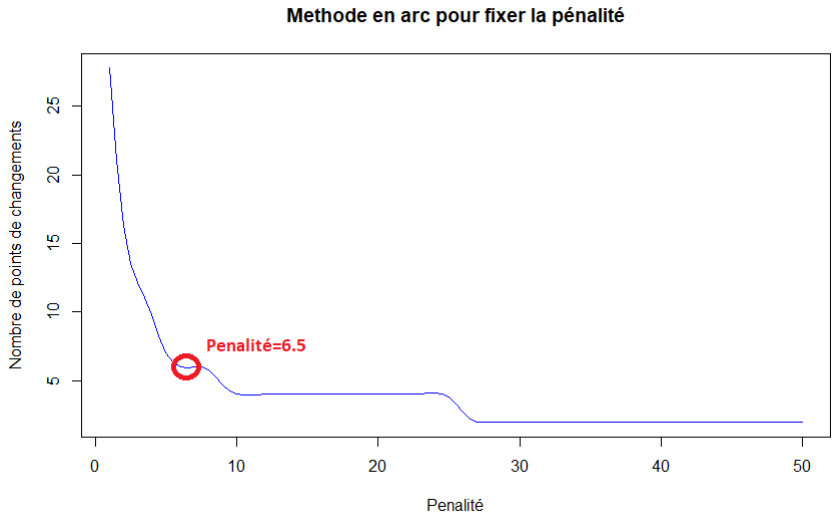


FIGURE 6.7 – Illustration de la méthode en Arc pour l'identification de la valeur de pénalité approprié à appliquer. La valeur prise ici, correspondant au coude de l'arc, est définie par le petit rond rouge sur le graphique.

De manière empirique, nous avons défini comme intervalle d'exploration utilisé pour la valeur de pénalité l'intervalle]0, 25] avec un incrément (pas) de 0,5. Cet intervalle permet surtout une grande flexibilité par rapport à l'hétérogénéité des traces énergétiques pouvant être analysées. Autrement, cela peut être vu comme le pire des cas. Ainsi, cet intervalle peut être dans une part significative des cas réduit. Malgré l'espace d'exploration des valeurs des pénalités, le temps de calcul de la méthode de recherche en arc se voit certes augmenté, mais reste tout de même relativement raisonnable. Cela grâce notamment, à la vitesse de calcul de l'algorithme PELT et à la possibilité de facilement paralléliser le traitement et la recherche dans l'intervalle d'exploration.

6.5.6 Détails techniques

La précédente étape de détection de points de changement permet d'identifier la segmentation/partitionnement optimale d'une trace énergétique. Néanmoins cela ne suffit pas car l'identification des états énergétiques reste encore abstraite sans liaison avec le code embarqué exécuté. Cette relation est assurée par notre processus d'inférence et notre framework de cartographie. En effet, via celui-ci permet de garder la relation entre un état énergétique détecté sur la trace et la fonction logicielle ayant induit cet état.

De plus, certains composants périphériques ne peuvent pas être mesurés individuellement. Cela est le cas, par exemple, pour un composant soudé directement sur la plate-forme matérielle du système embarqué. Nous résolvons ce problème en utilisant un état énergétique de base dans notre mesure. Ainsi, si un composant périphérique est mesuré individuellement, cet état de base est nul. Dans le cas contraire, celui-ci devient égal à la consommation moyenne de la plate-forme matérielle, dans un état d'activité IDLE (i.e. état dans lequel la plate-forme matérielle n'effectue aucune activité logicielle effective).

Enfin, certains composants périphériques peuvent exposer un schéma de consommation énergétique dit asynchrone. Autrement dit, l'énergie est encore physiquement et effectivement consommée après l'exécution de la fonction logicielle. Pour ces types de composants, nous mettons en œuvre une idée simple. Celle-ci affirmant que l'effet (i.e. consommation ou impact) énergétique d'une fonction est terminé lorsque la consommation dénotée sur la trace énergétique revient à un état similaire à celui précédant l'exécution de la fonction.

6.6 Regroupement des états et génération des automates énergétiques

Après l'exécution de la phase de détection d'état, ceci représente la dernière étape à effectuer. En effet, afin de réduire le nombre d'état trouvé et réussir à fournir au développeur une certaine vue synthétique et compréhensive, nous proposons de regrouper les états énergétique similaire. Cela a pour effet de former des états globaux résumant mieux les états énergétiques existants.

6.6.1 Types de regroupement

Nous proposons deux type de regroupement. Le premier regroupement concerne les états de puissance électrique, que nous noterons Ps. Ces états sont uniquement définis par la puissance électrique d'un état détecté. Autrement dit, il le sont par, le courant moyen d'un état et la tension fixe alimentant la plate-forme (i.e. puissance = courant moyen * tension). Le second groupement est lié aux états d'énergie (Es) qui sont définis, en outre, par la durée de l'état (c'est-à-dire Énergie de l'état = Puissance de l'état * durée de l'état). De plus, les états

d'énergie (Es) n'ont de signification que pour *"Fonctions actives"*. À la fin, chaque état d'alimentation du modèle énergétique basé sur un FSM est augmenté avec les états énergétiques correspondants.

L'objectif d'avoir de tels regroupements est de pouvoir fournir deux visions différentes qui sont toute les deux importantes pour un développeur de logiciels embarqués contraints en énergie. La vision de groupe d'état d'énergie permet d'obtenir une vision de la consommation énergétique d'opérations ou fonctions bornées dans le temps. De ce fait, Celle-ci ont un même temps de complétion fixe à travers plusieurs exécutions successives avec les même arguments d'entrées.

Ces arguments sont la raison de la présence du second regroupement concernant les états de puissance électrique. En effet, pour plusieurs types d'opérations (e.g. le cas d'utilisation LoRa décrit plus bas), la consommation énergétique va être étroitement liée au niveau de puissance de l'état. Les arguments d'entrées de l'opération exécutée définiront la durée d'étalement cette puissance. Cela vaut aussi pour des opérations qui peuvent être quasi-infinies jusqu'à intervention d'une autre opération (e.g. Mise en écoute d'un module de communication). Ainsi, avoir une vue qui ne montre que la puissance électrique permet au développeur de pouvoir construire par la suite des sous-modèles pouvant résumer le comportement interne d'une opération/fonction. Nous laissons cette tâche pour le développeur ou analyste, notre méthode d'inférence permettant justement de faciliter la mise en place de banc de test. Il est bien-sûr possible pour certaine type d'opération de mettre en place un simple modèle à régression linéaire. Cependant un tel modèle ne serait pas toujours adéquat selon la fonction analysée. C'est pour cette raison que nous préférons rester dans un stade où on fournit au développeur deux types de regroupement lui permettant de déjà avoir une vue globale du comportement d'une fonction.

6.6.2 Regroupement à travers la classification

Le processus d'inférence à cette étape doit regrouper tout les états de puissance, puis tout les états énergétiques similaires dans le but de former des états globaux plus discriminants. il existe plusieurs façon plus ou moins complexe afin de réaliser cette tâche. Dans un soucis de simplicité du processus d'inférence, nous avons choisi d'accomplir cette tâche de regroupement par le biais d'une méthode de classification.

Nous utilisons une approche de classification hiérarchique. Ce type d'approche a pour but la création de regroupements hiérarchiques pour un ensemble de points quelconque. Plus précisément nous concernant, Nous nous basons sur une approche de regroupement hiérarchique ascendante (HCA) décrite dans [**?**]. Cette méthode de classification est caractérisé par une simplicité de mise en œuvre et d'utilisation ainsi qu'une bonne efficacité.

Cette approche classifie un ensemble de points, appartenant à $\mathbb{R}$, en fonction des distances (très souvent distance euclidienne) séparant chaque points, formant ainsi des classes (*cluster*). La méthode effectue les regroupement en prenant les deux points les plus proches à chaque tour. Un nouveau point est créé et représente la distance moyenne entre les deux points du groupe. Au prochain tour, l'approche sélectionnera aussi deux nouveau points parmi le nouveau groupe de points existant. Ces itérations continuent avec des regroupements réalisés de manière ascendante jusqu'à formation d'un unique cluster. En même temps, un arbre nommé dendrogramme est construit (Figure 6.8). Celui-ci est peuplé avec tous les groupements effectués à chaque itération de l'approche HCA.

Afin de récupérer les clusters (groupes) formés, l'arbre de dendrogramme complètement construit doit être coupé à un certain niveau (Comme illustré en exemple en Figure 6.8). Cela peut être fait manuellement, mais dans

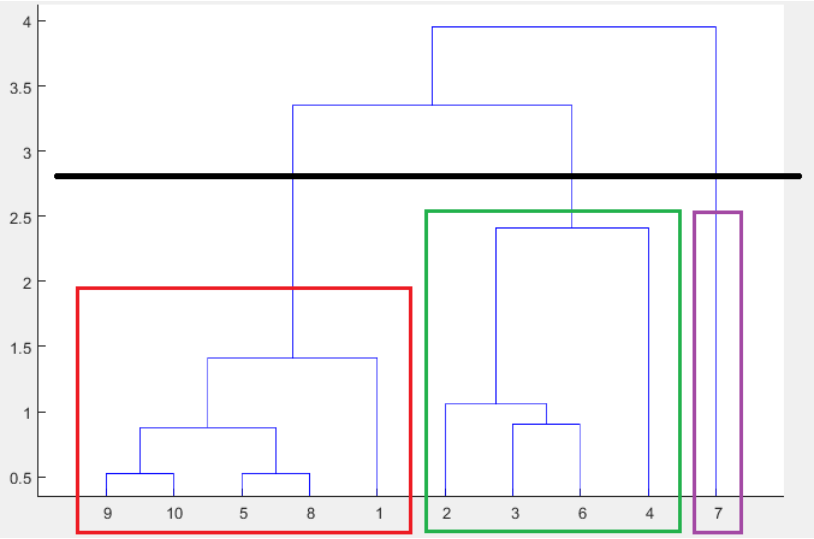


FIGURE 6.8 – Exemple d'un dendrogramme coupé au niveau de la ligne noir, donnant en résultat 3 clusters (rouge, vert, violet)

le but obtenir un processus automatisé sans interventions récurrentes du développeur, nous décidons d'utiliser une approche automatisé. Dans un premier temps, le critère de formation des cluster est modifié afin de remplacer la distance euclidienne par le critère de Ward [Jr.63]. Ce critère est implémenté dans le paquet R *FactoMineR*. Au lieu d'une simple distance métrique, il se base sur l'analyse de la variance afin de choisir les nouveaux clusters. Ce critère est pour cela aussi connu sous le nom du critère de variance minimum de Ward (ang. *Ward's minimum variance*). Deux métriques sont ainsi utilisées par ce critère : i) l'inertie intraclasse (la variance à l'intérieur d'un cluster) ; ii) l'inertie interclasse (la variance entres les clusters). L'addition de ces deux métrique résulte en l'inertie totale de la distribution.

La méthode de Ward consiste à chaque étape, en l'agrégation de deux clusters tel que l'augmentation de l'inertie intraclasse soit minimal (i.e. la minimisation de la réduction de l'inertie interclasse). Le dendrogramme ainsi construit via cette méthode est indexé par le gain de l'inertie interclasse à chaque étape. Afin de choisir la découpe adéquate du dendrogramme et donc le nombre de clusters, ce gain est justement utilisé par le package *FactoMineR*. Brièvement, la règle utilisée suggèrent une division en Q clusters lorsque l'augmentation de l'inertie interclasse entre $Q-1$ clusters et Q cluster est "bien plus grande" que celle entre Q cluster et $Q+1$ clusters. Formellement, on défini, $\Delta(Q)$ l'augmentation de l'inertie interclasse entre $Q-1$ clusters et Q. Le critère proposé est : $\frac{\Delta(Q)}{\Delta(Q+1)}$. Le nombre de cluster Q qui minimise ce critère est utilisé par l'approche pour la division du dendrogramme. La valeur de critère peut aussi être mis manuellement afin de privilégier un fort regroupement et peu de clusters au final ou au contraire un faible regroupement et beaucoup de clusters.

Après la formation des clusters d'états de puissance et d'états d'énergie, nous tirons parti de notre framework de cartographie énergétique. Celui-ci est utilisé afin de retrouver les fonctions qui ont induit un état de puissance/énergétique, ainsi que pour localiser dans quel cluster cet état est situé. En suivant la même opération par rapport à tout les états nous arrivons à former un modèle énergétique liant l'exécution d'une fonction, ses paramétrés ainsi que l'état énergétique et de puissance associés.

6.7 Exemples d'application

Nous avons décrit dans ce qui précède, nous avons décrit les motivations, le fonctionnement ainsi que schéma interne de notre processus d'inférence. Nous présentons dans cette derrière section deux exemples d'application pour les quelles notre méthodologie et notre processus d'inférence présentés dans ce chapitre, peut être d'un fort support au développeur d'applications embarquées.

6.7.1 Exemple 1 : Communication de type LoRa

Nous illustrons d'abord un exemple de fonctionnement de notre processus sur un périphérique embarqué de communication. Celui-ci est un module de communication longue-portée basé sur la technologie LoRa [SEMb] et plus précisément un puce SX1272 [SEMa] du module ATIM ARM-NANO-N8-LW [Ati]totalement adressable avec un pilote logicielle utilisant une simple interface série (UART). La téchnologie de communication LoRa opère sur la bande de fréquence des 868 MHz en Europe. Différentes largeurs de bande sont aussi possible allant de 7.8 khZ jusqu'à 500 khZ. Elle dispose d'une bande passante maximal de 50 kbps ainsi qu'une portée de communication allant jusqu'à 20 km au maximum.

L'application de notre processus d'inférence commence par la création assistée de benchmarks. Dans cet exemple nous utilisons le listing d'annotations suivant afin d'explorer l'impact énergétique de deux configurations différentes du module de communication. Les configurations représentent la puissance d'émission caractérisant le module de communication LoRa lors de l'émission de trame. Par ailleurs, nous explorons cet impact énergétique en explorant la transmission de trames de différentes tailles allant de 1 octets jusqu'à 251 octets (avec un pas de 50 octets).

```
/** !MODELGEN! {
 "id":        "LoRa_Comm",
 "funcType":  "INIT",
}} */
void lora_on();

/** !MODELGEN! {
 "id":        "LoRa_Comm",
 "funcType":  "DEINIT",
}} */
void lora_off();

/** !MODELGEN! {
 "id":        "LoRa_Comm",
 "funcType":  "ACTIVE",
 "prevFuncs": [],
 "nextFuncs": [],
 "args": {
  "1": {"type":"uint8_t*", "act":"CREATE", "size":300},
  "2": {"value":{"inf":1, "max":251, "step":50}}
}} */
void lora_send(uint8_t* buf, uint8_t size);

/** !MODELGEN! {
 "id":       "LoRa_Comm",
 "funcType": "CONF",
 "args": {
  "1": {"value":[POWER_HIGH, POWER_LOW]}
}} */
void lora_setPower(pwr_t power);
```

L'annotation se veut simple, le développeur annote ici deux fonctions du pilote permettant de commander le module de communication LoRa : la fonction d'envoi de trame "*lora_send*" qui est une fonction de type "ACTIVE", et la fonction "*lora_setPower*" qui est une fonction de type "CONF"(configuration) permettant de spécifier la puissance d'émission. L'annotation de la première fonction permet d'indiquer la création d'un tampon de transmission de trames, ainsi que l'espace d'arguments sur lequel il faudra itérer dans le benchmark généré. La seconde annotation permet quant à elle la définition d'une fonction comme étant une fonction de configuration modifiant donc l'état interne du module de communication LoRa. Ces opérations de transmission sont respectivement précédées et suivies par une initialisation du module "*lora_on*" et une dé-initialisation de celui-ci "*lora_off*".

Ces annotations génèrent donc deux benchmarks distincts. Un pour chaque configuration spécifiée, avec une itération sur l'espace d'argument des taille de trames à envoyer. Ces deux deux benchmarks passe donc par le framework de cartographie énergétique afin d'être profilés sur un plan énergétique. Les deux traces de mesure énergétique résultantes sont illustrées ci-dessous. En Figure 6.9, est décrit la consommation énergétique lors de l'utilisation d'une puissance de transmission faible pour la transmission de trames. Alors qu'en Figure 6.10 est illustrée la consommation lors de l'utilisation d'une puissance de transmission haute.

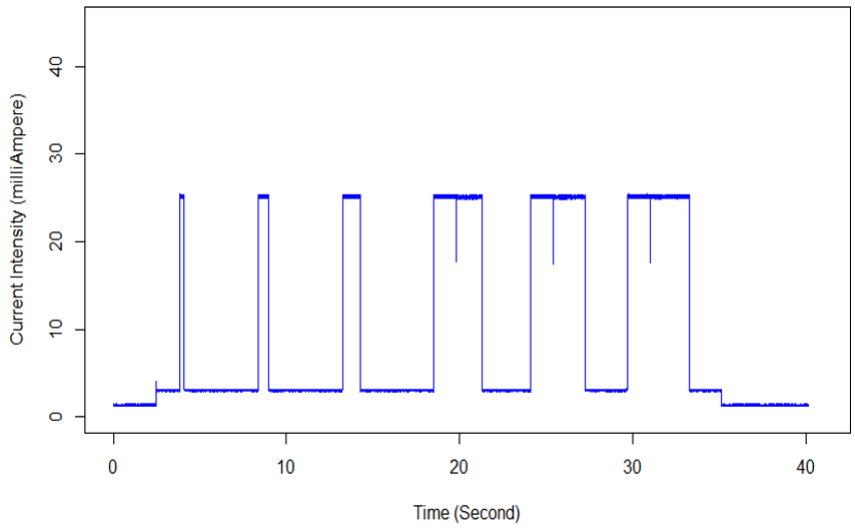


FIGURE 6.9 – Trace énergétique de transmission de trames via un module LoRa avec une faible puissance. Les trames sont de tailles : 1, 51, 101, 151, 201, 251 octets.

Les traces énergétiques produites suivent le processus d'inférence afin d'arriver au niveau de l'étape de détection d'état via l'approche PELT. Les deux illustrations 6.11 et 6.12 décrivent le résultat de cette détection. Comme nous pouvons facilement l'apercevoir les différents changement d'intensité du courant et donc de puissance sont délimités par l'exécution de l'approche PELT sur les traces énergétiques.

Cette détection permets ainsi de délimiter les états énergétique et de puissance sur chaque trace énergétique. Une fois cela, l'étape de regroupement d'état s'exécute afin réduire le nombre d'état existant et former ainsi des

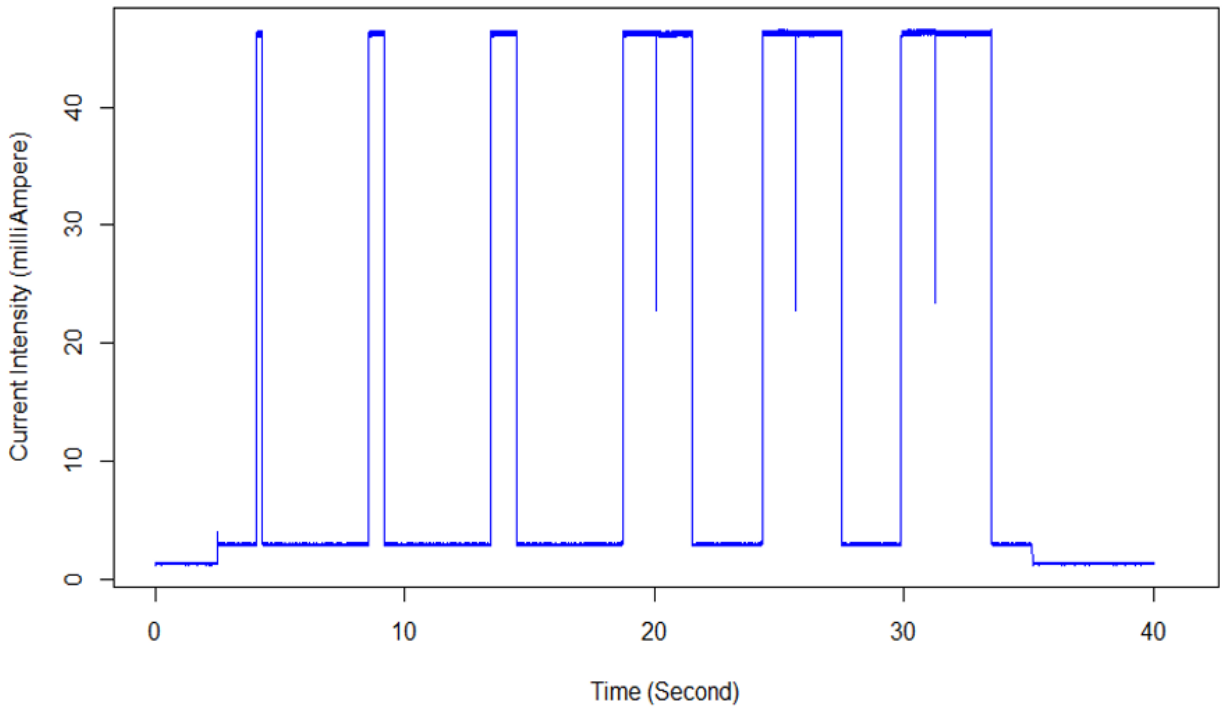


FIGURE 6.10 – Trace énergétique de transmission de trames via un module LoRa avec une haute puissance.

cluster d'états énergétiques et d'états de puissance. Le résultat de cette étape est décrit sur les Figures 6.13 et 6.14.

Une fois tous les regroupements d'état effectués. Le framework de cartographie énergétique est mis à contribution afin de lier les états situés dans les clusters formés avec les fonctions ayant déclenché chacun de ces états. Les résultats sont enregistrés sous la forme d'un simple tableau de correspondance dont nous donnons une illustration en Figure 6.15 qui représente une conjonction d'automate à états finies. Comme nous pouvons le voir, cela est un résumé des différents benchamrks exécutés et profilés énergétiquement précédemment. Ainsi nous apercevons le passage entre différent état de puissance, par exemple, de l'état "veille profonde (P0)" vers l'état "inactif (P1)". C'est uniquement après cela que selon la configuration utilisée (ici une seule configuration concernant la puissance d'émission) qu'on passe vers soit l'état (P2) ou (P3). Ces états de puissance donne lieux à différent états énergétique différents résultat du regroupement effectué précédemment.

6.8 Synthèse, conclusion et perspectives

Nous avons présenté dans ce chapitre notre contribution concernant la proposition d'une méthodologie et processus d'inférence de modèles énergétiques. Ce processus permet au développeur d'applications embarquée d'avoir une documentation résumant le comportement énergétique d'un composant périphérique en fonction des primitives exposées par le pilote logiciel de celui-ci. Nous avons décrit les motivations, le schéma interne, ainsi

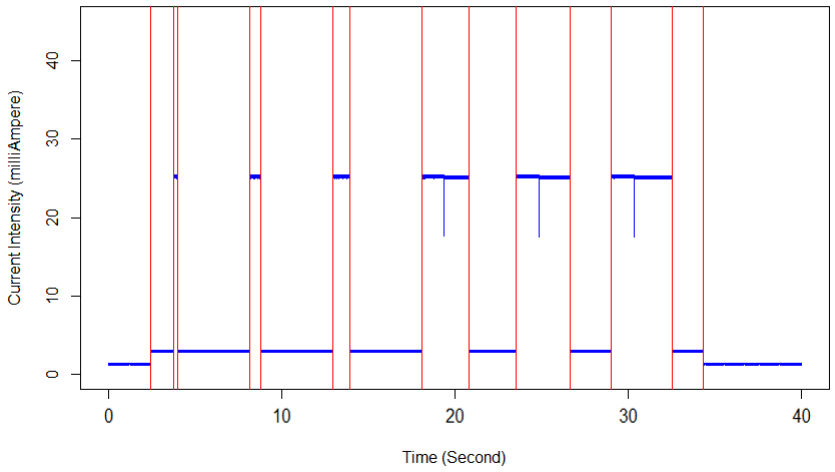


FIGURE 6.11 – Détection des états énergétique via l'approche PELT (configuration : puissance de transmission faible).

que le fonctionnement de ce processus sur des exemples réels. Ces derniers permettent de donner une preuve de concept de notre proposition. Bien-sûr plusieurs points sont encore à améliorer et à affiner afin de pousser vers une généralisation de ce procédé, notamment dans une vision plus industrielle.

Le premier point consiste en l'optimisation du générateur de benchmarks qui pour l'instant se concentre sur l'exploration des espaces d'arguments des fonctions annotées à inclure dans le benchmarks. A la demande du développeur, une liaison simple est faite entre les arguments en sortie d'une fonction vers les arguments d'entrées d'une fonction suivante. En effet, le manque criant de standardisation aux seins des pilotes logicielles des composants périphériques embarqués, complexifie grandement un processus de génération standard. Ainsi, une situation où chaque argument puisse être automatiquement lié avec d'autres arguments de fonctions quelque soit son niveau d'inclusion (structure, sous-structure, etc.) est encore utopiste. Une solution directe serait de mettre en place des règles de génération selon le composant utilisé. Néanmoins, Cela induirait un important effort de conception pour le générateur afin de reconnaître et d'apprendre un maximum de composants. Une solution plus simple représentant la perspective à explorer pour nos prochains travaux concerne une participation un peu plus forte du développeur au processus de génération. En fournissant à ce dernier une abstraction devant être implémentée par dessus la couche du pilote logicielle, il serait envisageable d'arriver à une situation générique quels que soient le composant périphérique ciblé.

Un autre aspect pouvant être intégré comme perspective proche, est l'aide du développeur dans l'utilisation des modèle fournis en tant que documentation énergétique. En effet, un support de comparaison et de mise en opposition de différents modèles générés peut être proposé afin d'aider directement et automatiquement le développeur dans ses choix de composants à utiliser.

Malgré ces quelques points manquants, nous avons étudié et proposé une solution sur un aspect longtemps délaissé par l'état de l'art, celui des composants périphériques. En effet, les avancés technologiques en matière

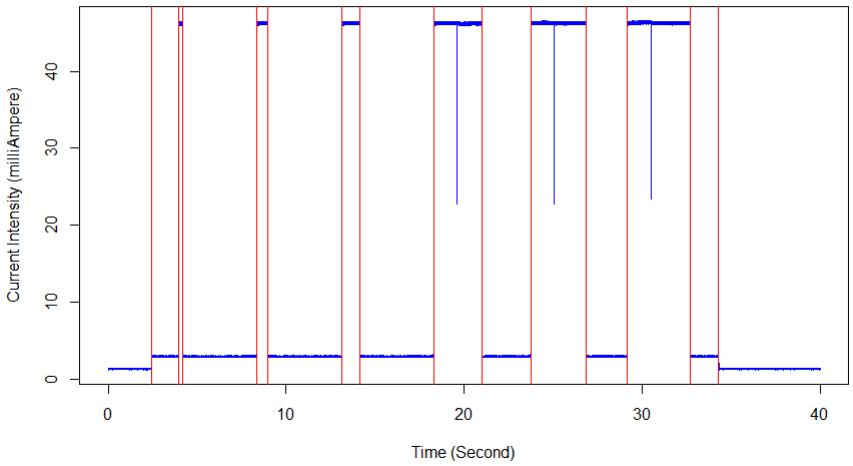


FIGURE 6.12 – Détection des états énergétique via l'approche PELT (configuration : puissance de transmission haute).

de fabrication de circuit qui ont fait évoluer le cadre de l'Internet des Objets font que le focus sur l'unique consommation énergétique du processeur (microcontrôleur) ne tiens plus d'actualité. En effet, nous sommes convaincu que l'étude de la consommation d'un objet connecté passe par l'analyse de la consommation du microcontrôleur mais aussi et surtout par celles des composants périphériques, ainsi que la manière dont ils sont utilisés par le développeur (i.e. le logiciel embarqué). Les modèles générés ici ont donc pour but de fournir ces informations au développeur d'applications embarquées. Même s'il n'ont qu'une valeur informative pour l'instant, des comparaisons automatiques peuvent être faites comme cités plus haut. Aussi, pour aller plus loin dans une perspective bien plus intéressante et productive d'un point du vue industriel. Les modèles peuvent être utilisés afin d'effectuer automatiquement une correction du code source de l'application embarquée. Ainsi, selon le comportement modélisé du périphérique, un arbitre pourrait automatiquement suggérer au développeur des améliorations du logiciel, ou encore faire les correction automatiquement et tester par ailleurs le bénéfice énergétique de cette correction.

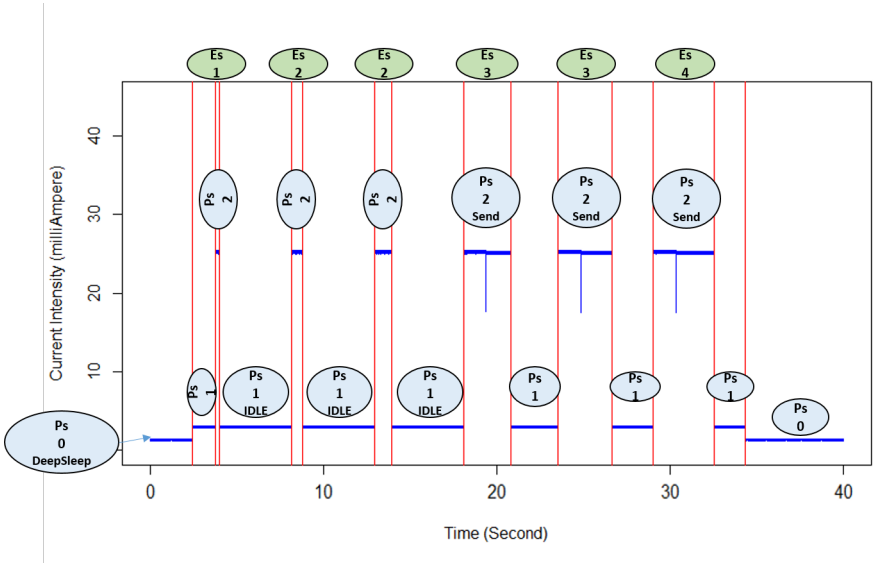


FIGURE 6.13 – Assignation et regroupement des états énergétiques et des états de puissance (configuration : puissance de transmission haute).

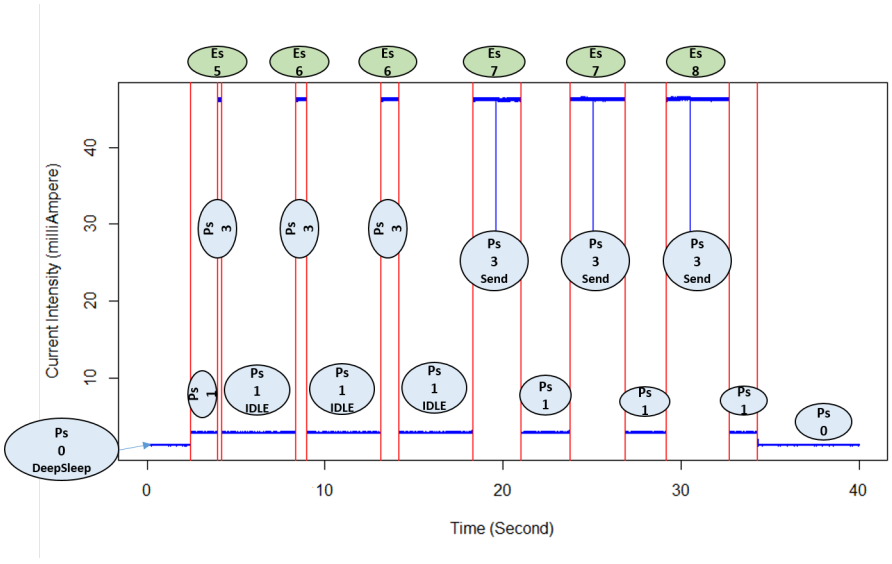


FIGURE 6.14 – Assignation et regroupement des états énergétiques et des états de puissance (configuration : puissance de transmission haute).

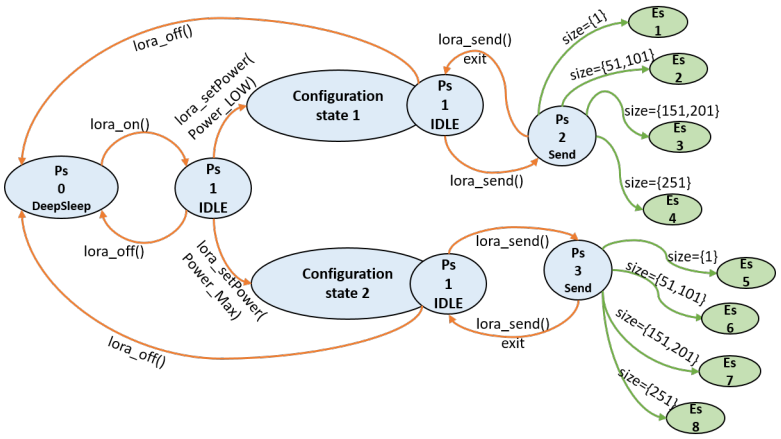


FIGURE 6.15 – Modèle énergétique obtenu sur le cas d'utilisation de transmission de trames de différentes tailles.

Quatrième partie

Conclusion et perspectives

Chapitre 7

Conclusion et perspectives

"La science n'atteint jamais son but parce que le but n'en finit pas de se dérober
- et qu'en vérité il n'y a pas de but : la science est une tâche infinie.
Sa grandeur est de se présenter comme un rêve toujours inassouvi."
Jean d'Ormesson, C'est une chose étrange à la fin que le monde

7.1 Synthèse

7.2 Résumé des contributions

7.3 Conclusion

7.4 Perspectives

7.5 Conclusion personnelle

7.6 Conclusion critique

7.7 Conclusion méthodologique

Annexes

Bibliographie

[AG02] Elena Andreou and Eric Ghysels. Detecting multiple breaks in financial market volatility dynamics. *Journal of Applied Econometrics*, 17(5) :579–600, 2002.

[Age14] International Energy Agency. More data, less energy : Making network standby more efficient in billions of connected devices. Technical report, 2014.

[AH09] J. Andersen and M.T. Hansen. Energy bucket : A tool for power profiling and debugging of sensor nodes. In *Third International Conference on Sensor Technologies and Applications, 2009. SENSORCOMM '09*, pages 132–138, 2009.

[AIL99] Reena Aggarwal, Carla Inclan, and Ricardo Leal. Volatility in emerging stock markets. *Journal of Financial and Quantitative Analysis*, 34(1) :33–55, 1999.

[AJ03] Eitan Altman and Tania Jiménez. Novel delayed ack techniques for improving tcp performance in multihop wireless networks. 2775 :237–250, 2003.

[AL89] Ivan E. Auger and Charles E. Lawrence. Algorithms for the optimal identification of segment neighborhoods. *Bulletin of Mathematical Biology*, 51(1) :39–54, Jan 1989.

[Ati] Atim. Arm-nano arm-n8-lw lora.

[BBM00] J. V. Braun, R. K. Braun, and H. G. Muller. Multiple changepoint fitting via quasilikelihood, with application to dna sequence segmentation. *Biometrika*, 87(2) :301–314, 2000.

[BBV09] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones : A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[BGK+06] Brendan Burns, Kevin Grimaldi, Alexander Kostadinov, Emery D. Berger, and Mark D. Corner. Flux : A language for programming high-performance servers. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 13–13, Berkeley, CA, USA, 2006. USENIX Association.

[Bra89] R. Braden. Requirements for internet hosts – communication layers. Technical report, 10 1989. RFC 1122.

[BYAH06] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac : A short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th International*

Conference on Embedded Networked Sensor Systems, SenSys '06, pages 307–320, New York, NY, USA, 2006. ACM.

[CBV+17] N. Cherifi, A. Boe, T. Vantroys, C. Herault, and G. Grimaud. Automatic inference of energy models for peripheral components in embedded systems. In *2017 5th International Conference on Future Internet of Things and Cloud*, pages 1–8, Aug 2017. doi : 10.1109/FiCloud.2017.53. url : https://hal.inria.fr/hal-01599169v1.

[CGBV16] N. Cherifi, G. Grimaud, A. Boe, and T. Vantroys. Toward energy profiling of connected embedded systems. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–4, Nov 2016. doi : 10.1109/NTMS.2016.7792483. url : https://hal.inria.fr/hal-01599164v1.

[CGVB15] N. Cherifi, G. Grimaud, T. Vantroys, and A. Boe. Energy consumption of networked embedded systems. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 639–644, Aug 2015. doi : 10.1109/FiCloud.2015.90. url : https://hal.inria.fr/hal-01193142v1.

[CGVB16] N. Cherifi, G. Grimaud, T. Vantroys, and A. Boe. Mesure de la consommation Énergétique des systèmes embarqués communicants. In *Journées scientifiques sur l'énergie et les radio-sciences*, Mar 2016. http://ursi-france.telecom-paristech.fr/fileadmin/journees_scient/docs_journees_2016/data/articles/000020.pdf.

[co] clang org. clang : a c language family frontend for llvm.

[Cor16] Atmel Corporation. Atmel power debugger, 2016.

[DEFT11] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes. Powertrace : Network-level power profiling for low-powerwireless networks. Technical report, SICS, 2011.

[Dev] Valgrind™ Developers. Callgrind : a call-graph generating cache and branch prediction profiler.

[DGV04] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462, Nov 2004.

[DMN10] W. Du, F. Mieyeville, and D. Navarro. Modeling energy consumption of wireless sensor networks by systemc. In *2010 Fifth International Conference on Systems and Networks Communications*, pages 94–98, Aug 2010.

[DÖTH07] Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 28–32, New York, NY, USA, 2007. ACM.

[Dun03] Adam Dunkels. Full TCP/IP for 8 Bit Architectures. In *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*, San Francisco, may 2003. usenix.

[Dun08] A. Dunkels. Sicslowpan - internet-connectivity for low-power radio systems. Technical report, SICS, 2008.

[Dun11] Adam Dunkels. The contikimac radio duty cycling protocol, 2011.

[DVA04] Adam Dunkels, Thiemo Voigt, and Juan Alonso. Making TCP/IP Viable for Wireless Sensor Networks. In *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN 2004), work-in-progress session*, Berlin, Germany, January 2004.

[EL05] Carla S. Ellis and R. Lebeck. Experiences in managing energy with ecosystem. *IEEE PerCom*, 2005.

[EÖV+09] Joakim Eriksson, Fredrik Österlind, Thiemo Voigt, Niclas Finne, Shahid Raza, Nicolas Tsiftes, and Adam Dunkels. Accurate power profiling of sensornets with the cooja/mspsim simulator. In *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems, MASS 2009, 12-15 October 2009, Macau (S.A.R.), China*, pages 1060–1061, 2009.

[Eri99] 3gpp Ericsson. Rrc protocol states. Technical report, 1999.

[FDLS08] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. Quanto : Tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 323–338, Berkeley, CA, USA, 2008. USENIX Association.

[Fer04] Viviana Fernandez. Detection of breakpoints in volatility. Documentos de Trabajo 194, Centro de Economía Aplicada, Universidad de Chile, 2004.

[FFF06] Nicolas Fournel, Antoine Fraboulet, and Paul Feautrier. Embedded Systems Energy Characterization using non-Intrusive Instrumentation. Technical report, LIP - ENS Lyon, October 2006. Research Report RR2006-37.

[FFF07] N. Fournel, A. Fraboulet, and P. Feautrier. eSimu : a Fast and Accurate Energy Consumption Simulator for Real Embedded System. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pages 1–6, June 2007.

[FFF09] Nicolas Fournel, Antoine Fraboulet, and Paul Feautrier. Embedded software energy characterization : Using non-intrusive measures for application source code annotation. *Journal of Embedded Computing*, 3(3) :10, July 2009.

[FRP+06] Frank Hanns Paul Fitzek, S Rein, M.V Pedersen, Gian Paolo Perrucci, T Schneider, and C Gühmann. Low complex and power efficient text compressor for cellular and sensor networks. 2006.

[FS99] Jason Flinn and M. Satyanarayanan. Powerscope : A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99, pages 2–, Washington, DC, USA, 1999. IEEE Computer Society.

[Gar17] Gartner. Leading the iot, 2017.

[GMS14] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva. Network-layer security for the internet of things using tinyos and blip. *International Journal of Communication Systems*, 27(10) :1938–1963, 2014.

[HC08] Jonathan W. Hui and David E. Culler. Ip is dead, long live ip for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 15–28, New York, NY, USA, 2008. ACM.

[Hin70] David V. Hinkley. Inference about the change-point in a sequence of random variables. *Biometrika*, 57(1) :1–17, 1970.

[HJE+14] Timo Hönig, Heiko Janker, Christopher Eibel, Wolfgang Schröder-Preikschat, Oliver Mihelic, and Rüdiger Kapitza. Proactive Energy-aware Programming with PEEK. In *Proceedings of the 2014 International Conference on Timely Results in Operating Systems*, TRIOS'14, Berkeley, USA, 2014. USENIX Association.

[Hol13] Simon Hollis. Mageec project - power measurement board, 2013.

[HSL+16] Anwar Hithnawi, Hossein Shafagh, Su Li, James Gross, and Simon Duquennoy. CrossZig : Combating Cross-Technology Interference in Low-power Wireless Networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN 2016)*, Vienna, Austria, apr 2016.

[Ins17] Texas Instruments. Cc2420 : 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver. Technical report, 2017.

[Int12] Maxim Integrated. *MAX4378TAUD+ : High-Side Current-Sense Amplifiers with Internal Gain*, 2012.

[ISy15] ISystem. Arm cortex on-chip trace, 2015.

[JDCS07a] Xiaofan Jiang, P. Dutta, D. Culler, and I. Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *6th International Symposium on Information Processing in Sensor Networks, 2007. IPSN 2007*, pages 186–195, 2007.

[JDCS07b] Xiaofan Jiang, Prabal Dutta, David Culler, and Ion Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, pages 186–195, New York, NY, USA, 2007. ACM.

[JG87] Tang Jen and A.K. Gupta. On testing homogeneity of variances for gaussian models. *Journal of Statistical Computation and Simulation*, 27(2) :155–173, 1987.

[Jr.63] Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301) :236–244, 1963.

[KAEEJ10] Rebecca Killick, Idris A Eckley, Kevin Ewans, and Philip Jonathan. Detection of changes in the characteristics of oceanographic time-series using changepoint analysis. 09 2010.

[KFE12] R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500) :1590–1598, 2012.

[LF14] Anders Lundgren and Lotta Frimanson. I-jet iar systems datasheet, 2014.

[LFZ+13] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. Flocklab : A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, IPSN '13, pages 153–166, New York, NY, USA, 2013. ACM.

[Lim11] ARM Limited. Embedded trace macrocell architecture specification, 2011.

[LLWC03] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM : Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA, 2003. ACM.

[LMGF13] Q. Li, M. Martins, O. Gnawali, and R. Fonseca. On the effectiveness of energy metering on every node. In *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 231–240, May 2013.

[LMP+05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *TinyOS : An Operating System for Sensor Networks*, pages 115–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Lou07] J. T. Louhi. Energy efficiency of modern cellular base stations. In *INTELEC 07 - 29th International Telecommunications Energy Conference*, pages 475–476, Sept 2007.

[LWG05] O. Landsiedel, K. Wehrle, and S. Gotz. Accurate prediction of power consumption in sensor networks. In *The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, pages 37–44, 2005.

[MB06] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 403–414, New York, NY, USA, 2006. ACM.

[MHY+06] D. Mclntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. J. Kaiser. The low power energy aware processing (leap) embedded networked sensor system. In *2006 5th International Conference on Information Processing in Sensor Networks*, pages 449–457, 2006.

[ML08] David Moss and Philip Levis. Box-macs : Exploiting physical and link layer boundaries in lowpower networking. Technical report, 2008.

[mot06] moteiv. Tmote sky : Ultra low power ieee 802.15.4 compliant wireless sensor module. Technical report, 2006.

[Mul07] Geoff Mulligan. The 6lowpan architecture. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 78–82, New York, NY, USA, 2007.

[NAJ12] Christopher F. H. Nam, John A. D. Aston, and Adam M. Johansen. Quantifying the uncertainty in change points. *Journal of Time Series Analysis*, 33(5) :807–823, 2012.

[ÖED10] Fredrik Österlind, Joakim Eriksson, and Adam Dunkels. Cooja TimeLine : A power visualizer for sensor network simulation. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 385–386. ACM, 2010.

[OVLW04] Adam B. Olshen, E. S. Venkatraman, Robert Lucito, and Michael Wigler. Circular binary segmentation for the analysis of array-based dna copy number data. *Biostatistics*, 5(4) :557–572, 2004.

[PBM+04] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. Atemu : a fine-grained sensor network simulator. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 145–152, Oct 2004.

[PHC04] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 95–107, New York, NY, USA, 2004. ACM.

[PHZ+11] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained Power Modeling for Smartphones Using System Call Tracing. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.

[PHZ12] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app ? : Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 29–42. ACM, 2012.

[PRL+05] Franck Picard, Stephane Robin, Marc Lavielle, Christian Vaisse, and Jean-Jacques Daudin. A statistical approach for array cgh data analysis. *BMC Bioinformatics*, 6(1) :27, Feb 2005.

[Rom88] J. Romkey. A nonstandard for transmission of ip datagrams over serial lines : Slip, 1988.

[RSV+05] H. Ritter, J. Schiller, T. Voigt, A. Dunkels, and J. Alonso. Experimental evaluation of lifetime bounds for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks*, 2005.

[SA07] K. M. Z. Shams and M. Ali. Wireless power transmission to a buried sensor in concrete. *IEEE Sensors Journal*, 7(12) :1573–1577, Dec 2007.

[SEMa] SEMTECH. Lora transceiver : Sx1276.

[SEMb] SEMTECH. What is lora ?

[SHCW04] Victor Shnayder, Mark Hempstead, Bor-Rong Chen, and Matt Welsh. PowerTOSSIM : Efficient Power Simulation for TinyOS Applications. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[SK74] A. J. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30(3) :507–512, 1974.

[SKG+07] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon : A Language and Runtime System for Perpetual Systems. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 161–174, New York, NY, USA, 2007. ACM.

[SKZS12] S. Schubert, D. Kostic, W. Zwaenepoel, and K.G. Shin. Profiling software for energy consumption. In *2012 IEEE International Conference on Green Computing and Communications (GreenCom)*, pages 515–522, 2012.

[SMK08] T. Stathopoulos, D. Mclntire, and W.J. Kaiser. The energy endoscope : Real-time detailed energy accounting for wireless sensor nodes. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 383–394, April 2008.

[Soc03] IEE Computer Society. 802.15.4 : Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). Technical report, 2003.

[Stm] Stmicroelectronics. Discovery kit with stm32f407vg mcu : User manual.

[STm14] STmicroelectronics. *Getting started with STemWin Library : Application Note*, 2014.

[STm17] STmicroelectronics. *Discovery kit for STM32F7 Series with STM32F746NG MCU - User manual - Rev4*, 2017.

[TIPV+16] V. Toldov, R. Igual-Pérez, R. Vyas, A. Boé, L. Clavier, and N. Mitton. Experimental evaluation of interference impact on the energy consumption in wireless sensor networks. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, June 2016.

[TLP05] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora : scalable sensor network simulation with precise timing. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 477–482, April 2005.

[VNT12] Ekhiotz Jon Vergara and Simin Nadjm-Tehrani. Energy-aware cross-layer burst buffering for wireless communication. In *Proceedings of the 3rd International Conference on Future Energy Systems : Where Energy, Computing and Communication Meet*, e-Energy '12, pages 24 :1–24 :10, New York, NY, USA, 2012. ACM.

[Wei] Josef Weidendorfer. Kcachegrind, call graph viewer.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3) :66–75, september 1991.

[Wes06] C. Westphal. Layered ip header compression for ip-enabled sensor networks. In *IEEE International Conference on Communications (ICC)*, volume 8, pages 3542–3547, June 2006.

[YKJ+12] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. AppScope : Application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, pages 36–36. USENIX Association, 2012.

[ZELV02] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. ECOSystem : Managing energy as a first class operating system resource. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pages 123–132. ACM, 2002.

[ZELV03] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy : A unifying abstraction for expressing energy management policies. In *In Proceedings of the USENIX Annual Technical Conference*, pages 43–56, 2003.

[ZX13] R. Zhou and G. Xing. Nemo : A high-fidelity noninvasive power meter system for wireless sensor networks. In *Information Processing in Sensor Networks (IPSN), 2013 ACM/IEEE International Conference on*, pages 141–152, April 2013.

Résumé

La désignation sous le terme d'Internet des Objets (ang. *Internet of Things, IoT*) regroupe un ensemble vaste de systèmes différents. Chaque objet connecté répond à un usage particulier. Il se compose d'un ensemble de capteurs permettant de récupérer des informations en lien avec ses différentes tâches. Ces informations peuvent être traitées localement ou transmises à des station distantes, provoquant la mise en action d'un ensemble d'actuateurs et de modules de communication.

Un nombre significatif de ces objets ne disposent pas d'une alimentation continue et fonctionnent donc grâce à des batteries. Par ailleurs, il existe de nombreux cas d'utilisation où le rechargement de la batterie se trouve être difficile voire impossible (e.g. objet enfoui dans le béton pour la surveillance de structures). De ce fait, l'aspect énergétique (stockage, utilisation et récupération) représente une contrainte primordiale à prendre en compte lors du développement de l'application embarquée sur l'objet. Les développeurs de logiciels pour systèmes embarqués sont les premiers confrontés à cette problématique. Ces derniers se retrouvent souvent à jongler entre le cahier des charges des fonctionnalités de l'objet, ses caractéristiques techniques et les restrictions énergétiques déterminant la durée de vie maximale de ce dernier. La problématique de ma thèse consiste à placer l'énergie comme objet de premier ordre pour les logiciels embarqués et pour cela fournir une assistance aux développeurs. L'objectif est que la prise en compte des critères énergétiques devienne un élément naturel pour le développeur. Ces travaux ont été menés dans le cadre d'une thèse CIFRE avec la société Worldline.

Je propose dans cette thèse une méthodologie et des outils pour soutenir les activités du développeurs. En outre, j'affirme que la capacité de mesurer et de suivre la consommation énergétique des objets connectés, puis de la corréler au logiciel sous-jacent permet d'améliorer l'efficacité énergétique globale en mettant en œuvre de bonnes pratiques liées à l'utilisation des différents composants matériels.

Dans cette thèse, j'étudie dans un premier temps l'état de l'art connexe aux approches de profilage énergétique d'un code source embarqué. Une taxonomie de ces approches est réalisée. Celles-ci sont étudiées, classées et discutées selon plusieurs critères discriminants. Ces critères sont : l'estimation de la durée de vie, la gestion de la consommation énergétique, la résolution du profilage énergétique, l'assistance au développeur et le coût de mise en œuvre. Suite à cette analyse, j'expérimente et mets en opposition sur un cas d'usage d'IoT concret, deux familles de méthodes de profilage énergétique. La première est basée sur la simulation de l'application embarquée. La deuxième estime la consommation énergétique en exécutant l'application dans son environnement de déploiement. Je démontre ainsi les faiblesses des simulateurs énergétiques et je modélise la différence séparant leur estimation de la consommation énergétique réelle en environnement effectif de déploiement [CGVB15, CGVB16].

Grâce à ce constat, je mets en évidence trois points fondamentaux représentant les piliers de ma proposition : i) la nécessité de profiler la consommation énergétique de l'application embarquée dans le monde réel, à l'aide d'une plate-forme matérielle de mesure qui permet d'obtenir des chiffres de consommation précis, ii) placer la consommation énergétique au centre des préoccupations en proposant un cycle de développement reposant sur cette contrainte, iii) abstraire et faciliter la tâche du développeur lors de la phase de profilage énergétique. Partant de ce postulat, je propose un framework de cartographie et de visualisation pour le profilage de la consommation énergétique d'applications embarquées [CGBV16]. Ce dernier, en utilisant un processus d'instrumentation du code source transparent au développeur, est capable d'attribuer à chaque fonction du code embarqué sa

consommation énergétique correspondante. De plus, une analyse statistique basique des courbes physique de consommation permet de limiter et atténuer la surcharge de l'instrumentation.

Dans un deuxième temps, je propose une étude et une production de guides de développement destinés aux développeurs embarquées lors de l'utilisation de modules radio cellulaire. En effet, le contexte industriel de cette thèse, l'importance de la communication dans l'IoT et les tendances actuelles du marché font des modules cellulaires le parfait exemple à prendre précisément en compte. Outre la fourniture de guides de développement, cette étude a mis en exergue la complexité du comportement énergétique de certains modules radio ainsi que l'importance d'en tenir compte lors du développement du logiciel. Ceci additionné à l'hétérogénéité des comportements énergétiques au sein des périphériques embarqués a donné lieu à ma dernière contribution.

Je propose un processus d'inférence de modèles énergétiques pour périphériques embarqués (i.e. capteurs, actuateurs, modules de communication) [CBV+17]. En mettant à profit le framework de cartographie énergétique, je montre que des modèles énergétiques peuvent être générés en minimisant l'intervention du développeur embarqué dans le processus. Ces modèles sont certes moins enrichis que ceux de la littérature, néanmoins ils restent léger et donc facilement régénérables. Ils permettent rapidement de connaître les modes de consommations de nouveaux composants. Ainsi, grâce à une analyse statistique et automatique des états énergétiques contenus dans des échantillons de code source de test, je montre que nous pouvons générer de manière simplifiée un modèle énergétique pour chaque périphérique différent du système. De plus, ces modèles étant basés sur l'interface de programmation applicative (API) de chaque pilote de périphérique, il est possible de contourner ainsi la limitation de non accès aux sources, existante dans la majorité du secteur industriel.

En conclusion, durant cette thèse, j'ai soulevé comme principal problématique, la complexité au regard du développeur de gérer la consommation énergétique de son application embarquée. Après un état de l'art et plusieurs expérimentations, j'ai proposé comme solution, un framework de cartographie de la consommation énergétique d'un logiciel embarqué. Celui-ci permet au développeur embarqué de mieux appréhender la consommation énergétique de son application dés la phase de conception. Enfin, j'ai proposé un processus d'inférence de modèles énergétiques pour périphériques embarqués. Ce processus permet au développeur d'être proactif lors du développement embarqué en ayant la possibilité de générer un modèle résumant le comportement d'un périphérique venant d'être intégré à la plate-forme embarqué cible.

Mots-clés: Informatique

Abstract

ToDo, lets translate the resume in english.

Keywords: Computer science

Main : *version du mardi 20 février 2024 à 17 h 17*