Middleware support for energy awareness in the Internet of Things (IoT)

Pedro Victor Borges Caldas da Silva

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAS016

Thèse de doctorat

Middleware support for energy awareness in the Internet of Things (IoT)

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Évry, le 12 Décembre 2022, par

PEDRO VICTOR BORGES CALDAS DA SILVA

Composition du Jury :

Philippe Roose Maître de Conférence, HDR, Université de Pau et des Pays de l'Adour	Rapporteur
Romain Rouvoy Professeur, Université de Lille	Rapporteur
Gordon Blair Professeur, Lancaster University	Examinateur
Anne Cécile Orgerie Directrice de Recherche CNRS, IRISA Rennes	Examinatrice
Sophie Chabridon Professeure, Télécom SudParis (SAMOVAR)	Examinatrice
Thais BATISTA Professeure, Universidade Federal do Rio Grande do Norte (UFRN)	Examinatrice
Chantal TACONET Maître de Conférence, HDR, Télécom SudParis (SAMOVAR)	Directrice de thèse

Abstract

The Internet of Things (IoT) is characterized by a myriad of geographically dispersed devices and software components as well as high heterogeneity in terms of hardware, data, and protocols. Over the last few years, IoT platforms have been used to provide a variety of services to applications such as device discovery, context management, and data analysis. However, the lack of standardization makes each IoT platform come with its abstractions, APIs, and interactions. As a consequence, programming the interactions between a consuming IoT application and an IoT platform is often time-consuming, error-prone, and depends on the developers' level of knowledge about the IoT platform. IoT middleware are proposed to alleviate such heterogeneity, provide relevant services, and ease application development.

As the energy efficiency of digital technology becomes a priority, the increase in IoT systems brings energy concerns. In this context, carefully designing interactions between IoT consumer applications and IoT systems with an energy-efficiency concern becomes essential. IoT middleware should not solely consider energy efficiency as a non-functional requirement. Instead, it needs to be at the solution's core as the middleware is expected to be shared by many applications and offer facilities to ease application development.

This work presents three contributions regarding energy-efficiency/awareness in IoT middleware for IoT consumer applications. The first contribution is the proposal of an IoT middleware for IoT consumer applications called IoTvar that abstracts IoT virtual sensors in IoT variables that are automatically updated by the middleware. The second contribution is the evaluation of the energy consumption of the interactions between IoT consumer applications and IoT platforms through the HTTP and MQTT protocols. This evaluation has led to the proposal of guidelines to improve energy efficiency when developing applications. The third contribution is the proposal of strategies for energy efficiency to be integrated into IoT middleware. Those strategies have been integrated into the IoTvar middleware to provide energy efficiency, but also energy awareness through an energy model and the management of

an energy budget driven by user requirements. The implementations of the IoT middleware architecture, with and without energy-efficiency strategies, have been evaluated, and the results show that we have a difference of up to 60% the energy used by IoT applications by applying strategies to reduce energy consumption at the middleware level.

Preface

This thesis is the result of my PhD research [1] under the supervision of Prof. Chantal TACONET and Prof. Thais BATISTA.

- Accepted publications

    1. *Mastering Interactions with Internet of Things Platforms through the IoTVar Middleware [Borg19]*

       Authors: Borges, Pedro Victor and Taconet, Chantal and Chabridon, Sophie and Conan, Denis and Batista, Thais and Cavalcante, Everton and Batista, Cesar

    2. *Analysis of the Impact of Interaction Patterns and IoT Protocols on Energy Consumption of IoT Consumer Applications[Cane22]* Authors: Borges, Pedro Victor and Taconet, Chantal and Canek, Rodrigo

    3. *Energy-awareness and energy efficiency in Internet of Things middleware: A systematic literature review [Accepted in Annals of Telecommunications]* Authors: Borges, Pedro Victor and Taconet, Chantal and Chabridon, Sophie and Conan, Denis and Batista, Thais and Cavalcante, Everton

- Under review publications

    1. *Taming Internet of Things Application Development with the IoTvar Middleware [Under Major Review in ACM TOIT]*

       Authors: Borges, Pedro Victor and Taconet, Chantal and Chabridon, Sophie and Conan, Denis and Batista, Thais and Cavalcante, Everton

[1] Funded by the "Futur & Ruptures" program from Institut Mines Télécom, Fondation, Fondation Mines-Télécom and Institut Carnot.

2. *Middleware supporting PIS: Requirements, solutions, and challenges (Book chapter in "Evolution of Pervasive Systems") [Under Press Release in Springer]*

   Authors: Taconet, Chantal and Batista, Thais and Borges, Pedro and Bouloukakis, Georgios, Cavalcante, Everton and Chabridon, Sophie and Conan, Denis and Desprats, Thierry and Muñante, Denisse

Figures

Tables

Support des middleware pour la prise en compte de la consommation énergétique dans l'Internet des objets

Résumé en français

L'Internet des objets (IoT) se caractérise par une myriade de dispositifs et de composants logiciels géographiquement dispersés ainsi que par une grande hétérogénéité en termes de matériel, de format de données et de protocoles. Au cours des dernières années, les plateformes IoT ont été proposées pour fournir une variété de services aux applications, tels que la découverte de dispositifs, la gestion du contexte et l'analyse des données. Cependant, le manque de standardisation fait que chaque plateforme IoT propose ses propres abstractions, API et patrons d'interactions. Par conséquent, la programmation des interactions entre une application IoT consommatrice de données et une plateforme IoT est complexe, sujette à des erreurs et demande un niveau de connaissance de la plateforme IoT approfondi de la part des développeurs. Les intergiciels IoT peuvent atténuer cette hétérogénéité, ils doivent fournir des services pertinents et ainsi faciliter le développement des applications.

L'efficacité énergétique de la technologie numérique devenant une priorité, l'augmentation du nombre de systèmes IoT pose des problèmes énergétiques. Dans ce contexte, il est essentiel de concevoir soigneusement les interactions entre les applications IoT grand public et les plateformes IoT en tenant compte de l'efficacité énergétique. Les intergiciels IoT ne doivent pas uniquement considérer l'efficacité énergétique comme une exigence non fonctionnelle laissée à l'application. Au contraire, parce qu'ils sont utilisés par de nombreuses applications, l'efficacité énergétique doit être au cœur de leur conception.

Cette thèse présente trois contributions concernant l'efficacité énergétique et la sensibilisation à l'énergie dans les intergiciels IoT pour les applications IoT consommatrices de données. La première contribution est la proposition d'un intergiciel IoT appelé IoTvar qui abstrait les capteurs virtuels IoT dans des variables IoT qui sont automatiquement mises à jour par l'intergiciel. La deuxième contribution est l'évaluation de la consommation d'énergie des interactions entre les applications IoT grand public et les plateformes IoT via les protocoles HTTP et MQTT. Cette évaluation a conduit à la proposition de lignes directrices pour améliorer l'efficacité énergétique des interactions. La troisième contribution est la proposition de stratégies d'efficacité énergétique pour des middleware IoT. Ces stratégies ont été intégrées dans l'intergiciel IoTvar pour assurer l'efficacité énergétique, mais aussi la sensibilisation à l'énergie par le biais d'un modèle énergétique et la gestion d'un budget énergétique fonction des exigences des utilisateurs. Les implémentations de l'architecture middleware IoT, avec et sans stratégie d'efficacité énergétique, ont été évaluées, et les résultats montrent que nous avons une diminution allant jusqu'à 60% de l'énergie consommée par les applications IoT en appliquant des stratégies pour réduire la consommation d'énergie au niveau du middleware.

Le document de thèse est composé de 7 chapitres couvrant une introduction, quelques éléments de contexte dans les domaines de l'Internet des objets et de la consommation d'énergie des logiciels, suivis d'une revue de l'état de l'art dans ces domaines, et de trois chapitres de contribution, avant de conclure et de donner des perspectives des travaux.

Dans le chapitre 1 (Introduction), nous motivons les défis liés à l'efficacité énergétique et à la prise de conscience énergétique dans les intergiciels pour l'IoT, puis nous présentons les quatre questions de recherche considérées dans la thèse. Ces questions de recherche portent sur i) la proposition par les intergiciels pour les applications consommatrices de données de l'IoT d'abstractions qui permettent l'interaction avec les plates-formes IoT, ii) le coût en termes de ressources informatiques de ces intergiciels, iii) l'impact sur la consommation d'énergie des protocoles IoT et des patrons d'interaction, et enfin iv) les stratégies visant à réduire la consommation d'énergie des applications IoT grand public.

Le chapitre 2 (Intergiciels pour l'IoT et efficacité énergétique) fournit ensuite le matériel de base nécessaire lié à la définition et aux architectures communément adoptées dans le domaine des systèmes IoT, intergiciels pour l'IoT et des plateformes IoT, ainsi que les notions liées à l'analyse de la consommation énergétique des logiciels. Ce chapitre examine trois plates-formes de gestion de contenu IoT, à savoir Orion, OM2M et muDEBS, afin d'étudier les fonctionnalités communément offertes et d'iden-

tifier les abstractions clés qui devraient être exposées aux applications consommatrices de l'IoT afin qu'elles interagissent avec un grand nombre de plateformes IoT. Ces fonctionnalités clés comprennent le modèle de données, les modèles d'interaction, les interfaces de programmation et les protocoles, la découverte et les capacités de filtrage. Au-delà de ces caractéristiques, ce chapitre présente les besoins énergétiques des applications IoT et examine les approches et les outils potentiels pour mesurer la consommation énergétique des logiciels. Cette étude conclut que les développements récents des plateformes IoT conduisent à la fourniture de solutions hétérogènes qui rendent l'effort de développement d'applications consommatrices de l'IoT difficile et sujet à des erreurs, rendant nécessaire la proposition d'intergiciels pour ces applications.

Le chapitre 3 (État de l'art concernant les middleware pour l'IoT et leur efficacité énergétique) présente un état de l'art en 3 volets : i) un examen systématique de la littérature sur l'efficacité énergétique des solutions middleware pour l'IoT, ii) les travaux connexes concernant l'efficacité énergétique des interactions (protocoles IoT consommatrices et patrons d'interaction) entre les applications consommatrices et les plateformes IoT, et enfin iii) les familles de middleware IoT pour les applications grand public. La revue de la littérature est plus spécifiquement couverte sous l'angle de l'efficacité énergétique et de la prise en charge de la sensibilisation à l'énergie des middleware IoT existants, qui est une exigence clé dans ce travail. La revue systématique a permis de recueillir et d'analyser 22 articles récents dans le domaine. Les études sélectionnées sont comparées en fonction des stratégies énergétiques qu'elles proposent pour i) l'adaptation en termes d'interactions et de protocoles, ii) le déplacement de tâches, iii) la sélection des nœuds IoT actifs, iv) l'apprentissage automatique concernant la mise à jour ded données et v) le filtrage des données. Le résultat de cet examen est que la plupart des études proposent au plus une stratégie pour réduire la consommation d'énergie, qui est mise en œuvre sous forme d'adaptation des interactions ou de déplacement des tâches dans la plupart des cas. Cependant, la plupart des études ne proposent aucune sensibilisation à l'énergie ou d'abstractions économes en énergie pour les applications IoT et, surtout, la plupart des études ont fourni des évaluations ciblées de la consommation d'énergie, manquant ainsi une évaluation plus systémique des économies globales réalisées par les solutions middleware fournies. Dans son deuxième volet, le chapitre souligne également le manque d'études empiriques sur la consommation d'énergie des protocoles HTTP et MQTT, qui sont largement adoptés par les applications consommatrices de l'IoT. Enfin, le chapitre motive l'adoption d'une approche middleware comme une approche plus flexible pour la mise en œuvre d'applications IoT, par rapport aux langages spécifiques au domaine ou aux approches mashup.

Les conclusions du chapitre 3 préparent le terrain pour la première contribution de la thèse présentée au chapitre 4 (IoTvar). IoTvar est un nouvel intergiciel qui propose des variables de type IoT et des proxys qui gèrent l'ensemble des interactions entre les plateformes IoT et les applications consommatrices de l'IoT. Plus précisément, IoTvar agit en tant que mandataire pour gérer la découverte des objets IoT, la mise à jour automatique des valeurs collectées (interactions, protocoles, présentation des données mises à jour, appel des listeners associés le cas échéant). Le coût de l'intégration d'une nouvelle plateforme IoT dans l'intergiciel est discuté et illustré sur les trois plateformes IoT sélectionnées au chapitre 2. Il est intéressant de noter qu'une grande partie du code traitant des modèles d'interaction et des protocoles peut être mise en œuvre indépendamment de ces plateformes, ce qui démontre un champ d'application plus large. Les abstractions d'intergiciel offertes par IoTvar sont prototypées dans le langage de programmation Java sous la forme d'un ensemble de classes et de primitives qui permettent à un développeur de déclarer facilement des variables IoT et d'enregistrer des listeners à déclencher pour traiter les données mises à jour le cas échéant. Par rapport au code écrit sans IoTvar, on peut observer que la quantité de code écrit est réduite d'un ordre de grandeur. En ce qui concerne l'impact sur la consommation d'énergie, on peut observer que IoTvar introduit une surcharge raisonnable qui peut être expliquée par le coût du mécanisme de proxy proposé.

Le chapitre 5 (Impact énergétique des patrons d'interaction et de protocoles IoT) approfondit la question de l'efficacité énergétique des protocoles IoT en réalisant une étude expérimentale et une comparaison des protocoles de réseau existants et des modèles d'interaction traditionnellement adoptés par les applications consommatrices IoT. Un protocole d'expérimentation est proposé et les résultats de l'expérimentation sont analysés. La combinaison de protocoles IoT (HTTP ou MQTT) et de modèles d'interaction (synchrone ou publication/souscription avec différentes exigences de qualité de service) est explorée et comparée dans différents contextes. Le chapitre conclut que le patron d'interaction publication/souscription est plus économe en énergie que le modèle d'interaction synchrone, et que le protocole MQTT surpasse HTTP de 20% en moyenne pour des interactions de type publication/souscription.

On peut également observer que le surcoût énergétique est proportionnel au niveau de qualité de service attendu, qui est corrélé au nombre de messages échangés. Enfin, l'impact de la charge utile d'un message bien que non négligeable montre que le nombre d'octets par joule progresse favorablement avec la taille de la charge utile. Cette analyse permet de fournir des indications intéressantes pour les concepteurs d'applications IoT, afin de réaliser d'importantes économies d'énergie. Ce résultat devrait guider le développement des futures applications IoT en proposant des modèles d'application qui adoptent de telles décisions par défaut.

Cette orientation est couverte par le chapitre 6 (Stratégies d'efficacité et de conscience énergétique dans IoTvar), qui traite de la mise en œuvre de diverses stratégies d'efficacité énergétique et de sensibilisation à l'énergie dans l'intergiciel IoTvar. Ces stratégies, à savoir le choix du protocole de communication, le regroupement de messages pour plusieurs variables, le changement de modèle d'interaction pour l'adaptation au temps de rafraîchissement et la gestion du budget énergétique, sont d'abord décrites, puis intégrées à l'intergiciel IoTvar, soit en tant que nouvelle couche énergétique, soit en tant qu'extensions de couches existantes. Il est intéressant de noter que l'intergiciel IoTvar est doté de la capacité de raisonner sur le coût énergétique des actions prises en charge pour ajuster sa configuration en fonction d'un budget énergétique donné. L'évaluation effectuée sur l'intergiciel IoTvar compare différentes configurations d'une application IoT par rapport à une base de référence sans stratégie. Alors que la version de base de l'intergiciel IoTvar présentait une surconsommation d'énergie par rapport aux plateformes IoT originales, les extensions d'efficacité énergétique et de sensibilisation d'IoTvar ont permis de réduire de manière significative la consommation d'énergie des applications IoT testées, jusqu'à 60%.

Enfin, le dernier chapitre propose une conclusion résumant les principales contributions apportées par ce travail et énumère quatre perspectives pertinentes pour ce travail, soulignant le potentiel d'extensions supplémentaires du middleware IoTvar pour prendre en charge les interfaces matérielles et réseau, mais aussi la généralisation des stratégies d'efficacité énergétique/sensibilisation proposées à un champ plus large de composants applicatifs, au-delà du champ des systèmes IoT.

Chapter 1

Introduction

1.1 Motivations

Nowadays the world is experiencing a huge surge of things connected to the internet known as the *Internet of Things (IoT)*. It is estimated that by the end of 2025 the world will be drawing near the mark of 100 billion IoT devices [Atti19, Okra15]. The *Internet of Things* (IoT) is a wide term referring to smart objects, services, and applications that collude to provide value-added content and services for end-users and systems.

The world is also experiencing a continuous increase need for the production of electricity, which has doubled in the last three decades of the past century [Bere07]. In 2012 the total energy consumption of Information and Communication Technology (ICT) reached 4.6% of the total worldwide electricity consumption, amounting to approximately 920 TWh [Lann13, VH14], and is estimated to account for more than 20% of global energy use by 2030 [Andr15]. The relentless increase in computing and communication requirements at a time when energy efficiency is a major concern for society is a paradox. In this context, the energy efficiency of digital technology becomes a priority.

The increase in IoT also brings energy concerns. The global energy consumption of IoT devices increases by 20% per year and it is estimated that it will represent about 46 TWh in 2025 [EAI16]. Thus, handling energy awareness and energy efficiency in the IoT context as first-class concepts is imperative for the Information Technology (IT) domain [Shai17b].

Internet of Things usually works based on small devices, and as explained by Benhamaid et al., *"IoT devices are usually small and battery limited and the exchange of massive information between these devices gives rise to enormous energy requirements. These requirements are often not supported by IoT devices*

and can quickly lead to battery depletion and the network's death." [Benh22]. That is why energy efficiency has firstly been taken into account in the design of software deployed on IoT devices [Muno19]. However, reducing software energy consumption in IoT may not be bounded only to IoT devices. It has been estimated that IoT devices generated around 67 zettabytes of data in 2020 [dbHF19]. Part of this volume of data is consumed by IoT applications. Thus, carefully designing interactions between IoT applications and IoT systems with an energy-efficiency concern is also essential.

Designing and implementing IoT applications is complex because it addresses different concerns: proper identification of various stakeholders' roles at the different phases of application development, heterogeneity in IoT systems, and handling of a vast amount of data from disparate devices [Pate15]. Furthermore, energy efficiency is a new requirement to be handled by IoT system designers. This gets even more challenging because developers still lack knowledge about software energy consumption [Pang16].

The complexity of developing IoT applications can be mitigated as developers can take advantage of IoT middleware that provides an abstraction layer between devices and developers, managing devices and protocols heterogeneity [Chaq12]. Middleware specialized for the IoT provides (i) abstractions for accessing volatile physical devices and managing the data produced by these devices, (ii) virtualization and aggregation functions of many devices into IoT systems, and (iii) interoperability patterns for managing software entities [Blai16, Ngu17, Razz16, Boul19]. The diverse role of middleware makes them suitable for integrating IoT applications with energy-efficient/aware solutions.

To facilitate the communication and data flow between IoT applications and devices, developers can also use IoT platforms [Mine16]. These platforms provide interfaces, interaction patterns, communication protocols, and computational capabilities to support the development of IoT applications. They also include middleware services that provide functionalities such as device discovery, context management, and data analysis [Ray16]. Context management IoT platforms, constitute a subset of IoT platforms that provide middleware components for accessing IoT data. Similarly, IoT consumer applications take advantage of the capabilities of both IoT platforms and middleware to access IoT data.

However, even with the support of IoT platforms and middleware, developing an IoT consumer application remains challenging. As an example, to display the current temperature at the Eiffel Tower in Paris, application developers have to program the interactions with an IoT platform that shows up

several virtual entities providing updated temperature data around the area. Even if most of the platforms provide similar features, developers need to learn, for each platform, specific APIs, data models, and communication protocols. Other development tasks include selecting and handling the appropriate interaction pattern (e.g., request/reply or publish/subscribe), (un) marshaling data, and manipulating sensor identifiers and metadata, such as quality attributes of sensor data. Furthermore, they may have to tune the frequency of interactions to limit the usage of computational resources.

Noureddine et al. [Nour13] highlight that middleware for distributed applications could contribute to optimizing or reducing the energy consumption of hardware devices, software services, and the platform itself. Thus, IoT middleware should not solely consider energy efficiency as a non-functional requirement. Instead, it needs to be at the solution's core as the middleware is expected to be shared by many applications and offer facilities to ease application development. IoT end-user applications need to work with green protocols and algorithms to ensure that the energy consumption is kept at only what is necessary for the expected functionalities. This idea becomes possible with the support of an energy-efficient middleware since it may provide the necessary services for such requirements.

In this context, it is also important to consider *energy-awareness*, i.e., understanding the energy consumption and how efficient this consumption is [Hass09]. With an energy-aware IoT middleware, an IoT application could know the energy efficiency of the application or even parts of the code itself. This facility contributes to easing the task of decreasing energy consumption. This capability is also relevant considering that application developers and users often have limited knowledge of how much energy their software consumes and which parts use the most energy [Pang16].

1.2 Research Questions

To be able to provide a solution for energy-efficiency/awareness in an IoT middleware some *Research Questions (RQ)* were raised. These RQs help to widen the view of the different aspects of IoT middleware and give an implemented, evaluated, and usable energy-efficient middleware for IoT application developers.

RQ1 : How can a middleware support common abstractions when interacting with IoT platforms?

  - IoT platforms provide services through specific API and data models, indicating a learning phase for developers. The answer to this research question will help conceive a solution usable by developers that hide the complexity of the underlying IoT platforms, leaving details of the implementation at the middleware level instead of the application level.

RQ2 : What is the cost, in terms of CPU usage, memory usage, and energy consumption, of abstracting IoT platforms using IoT middleware?

  - Having an abstraction to an IoT platform doing the processing of the information and communication comes at a cost. This RQ will give us answers regarding what is the overhead caused by the IoT middleware and how it can impact the resource consumption of an IoT application that chooses to use it.

RQ3 : What is the impact on the energy consumption of widely used protocols by IoT applications and with different interaction patterns?

  - This research question will help to understand the behavior of IoT protocols used by IoT applications. In a second step, it should provide guidelines to build energy-efficient strategies that rely on the features of the IoT protocols inside IoT middleware.

RQ4 : What are the strategies to be proposed by an IoT middleware to reduce the energy consumption of IoT consumer applications?

  - To provide energy-efficient/aware strategies to an IoT middleware, new components are added or modified in its architecture. These new components will provide the necessary implementation for the strategies and an evaluation is made to ensure a lower energy consumption by IoT applications using the middleware. This solution will give more credibility not only to the usage of the IoT middleware by IoT applications but also to provide future perspectives on the usage of certain energy-efficient strategies in other areas of IT.

1.3 Contributions

This work proposes three contributions regarding those RQs towards an energy-efficient/aware IoT middleware for IoT applications. These contributions start

with the conception of an IoT middleware, then we continue with the analysis of message patterns and IoT protocols to finally reach the confection of an energy-efficient/aware middleware that introduces strategies to lower the energy consumption at the application side.

The first contribution of this Ph.D. is **an IoT middleware (IoTvar) located between IoT-consuming applications and IoT platforms**. IoTvar proposes the paradigm of IoT variables able to manage all the interactions with virtualized sensors in IoT platforms. The interactions with the platform are managed by proxies on the application side. A generic architecture has been designed, the IoTvar middleware has been implemented and integrated with three IoT platforms: *FIWARE*, *OM2M*, and *MuDEBS*. We also report the results of experiments performed to evaluate IoTVar, showing that IoTVar reduces the effort required to declare and manage IoT variables. The experiments show the impact of IoTvar in terms of CPU, memory, and energy consumption.

The second contribution is **an experimental analysis of the energy consumption of the interactions between an IoT-consuming application and an IoT platform through two protocols proposed for interactions between an IoT application and IoT platforms: the HTTP and MQTT protocols**. For the HTTP protocol, we have studied both the publish-subscribe and the request-reply interaction patterns. For MQTT, we have studied the publish-subscribe interaction pattern with the three available Quality of Service. We also examine the impact of message payload on energy consumption.

The third contribution proposes **to integrate energy-efficiency/awareness strategies inside an IoT middleware such as the IoTvar middleware**. This is done by implementing new components inside the middleware that support the energy-efficient strategies as well as adding energy awareness to it. The components for energy efficiency/awareness are integrated into a new architecture of IoTvar and tests are done to gather data on the energy consumption of the middleware. The results gathered are then statistically analyzed, comparing the IoTVar middleware with and without the energy-efficient/aware mechanisms.

1.4 Outline of the document

The first part of the document presents the background and related works concerning IoT middleware and energy efficiency.

Chapter 2 provides the necessary vocabulary, definitions, and background informa-

tion concerning: (i) IoT middleware definition and architecture, (ii) IoT platform definition, architecture, and analysis, (iii) Analysis of measuring software energy consumption tools.

Chapter 3 analyses related works on three aspects, featuring a systematic literature review on energy efficiency in IoT middleware, works regarding energy efficiency in IoT protocols between IoT consumer applications and IoT platforms, and IoT middleware for consumer applications.

The second part of the document presents the contributions of the Ph.D.

Chapter 4 presents the architecture, implementation, and evaluation of the IoT-Var middleware, a middleware that provides abstractions for managing interactions between IoT consumer applications and IoT platforms. It provides the results regarding CPU, memory, and energy consumption.

Chapter 5 presents the results of experiments conducted to evaluate the impact of interaction patterns and IoT protocols on energy consumption of IoT consumer applications, and from those results draw some guidelines to build new interaction strategies.

Chapter 6 discusses the implementation of energy-efficiency/awareness strategies inside the IoTVar middleware and the results from an evaluation comparing an application using IoTVar with and without energy-efficiency/awareness strategies.

Finally, Chapter 7 presents the conclusion of the work along with future work to further enhance energy efficiency and energy awareness in IoT middleware.

Part I

IoT Middleware and Energy-Efficiency Background and Related Work

Chapter 2

IoT Middleware and Energy-Efficiency Background

This chapter provides the necessary vocabulary, definitions, and background information concerning energy-efficiency and IoT middleware. Section 2.1 details the IoT concept. Section 2.2 introduces the concepts and architecture of IoT middleware. Section 2.3 presents IoT platforms and Section 2.4 analyses common abstractions of context management IoT platforms with a comparison among three of them. Section 2.5 shows different tools for energy measurements, providing information on how they work and their purpose. Section 2.6 contains final conclusions about the background.

2.1 Internet of Things

2.1.1 IoT definitions

Internet of Things is the concept that aims to extend the regular Internet to real-world physical objects. In this paradigm, all the objects are either temporarily or permanently connected to a global network infrastructure. The IoT has been defined in different ways such as "The Internet of Things is a worldwide intelligent infrastructure which links things and related information" [Atzo10], "Things having identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environmental, and user contexts" [Infs08], and "a network of networks which enables the identification of digital entities and physical objects — whether inanimate (including plants) or animate (animals and human beings) — directly and without ambiguity, via standardized electronic identification systems and wireless mobile devices, and thus make it possible to retrieve, store, transfer and process data relating to them, with no discontinuity between the physical and virtual worlds" [Beng12].

The perception of the IoT has evolved by extending the fundamental definitions of this paradigm. Originally, the IoT was mainly thought of as the ability to automatically integrate into a system data related to the identity, location, and state of some physical entities. Recently, many authors characterize the IoT as a concept that encompasses a variety of technologies and research areas that aim to extend the current Internet to the real world.

2.1.2 IoT architecture

Figure 2.1 presents a classical IoT system distributed architecture such as those considered in this work. An IoT system consists of (1) IoT devices (sensors and actuators), (2) end-user applications (in this work, we focus on applications that consume sensor data that we call *IoT consumer applications*), (3) IoT platforms, gateways, and IoT middleware, standardized intermediates for interacting with IoT devices that deal with the high degree of hardware and software heterogeneity in IoT environments.

In the architecture, IoT applications are distributed on IoT devices (equipped with sensors and actuators) and end-user devices. Sensors are devices that detect external information (physical phenomena) and replace it with an electrical pulse that determines the measured value. Actuators are devices that convert electrical input into physical action. In this type of IoT

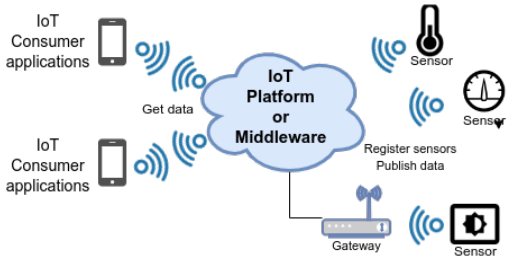


Figure 2.1. IoT distributed architecture

architecture, devices act autonomously and are interconnected to other nodes either directly with an IoT platform or through gateways. Both IoT platforms and gateways are connected to the Internet and make data available to end users' IoT applications.

2.1.3 IoT and context management

Nowadays, more and more applications consume high-level context information, obtained after processing, fusing, and filtering a large number of low-level context data collected from the user environment. This happens also because applications are becoming more distributed and able to consume more data. Villegas and Muller [Vill10] define context as follows: "Context is any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject that can affect the behavior of the system and/or its interaction with the user. This context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system's requirements, and maintained to support its dynamic evolution". The importance of the notion of context for pervasive computing has been identified in Coutaz et. al. [Cout05] which shows that context information can enrich human activities with new services able to adapt to the circumstances in which they are used.

The IoT impacts pervasive computing and context management because of the huge amount of data becoming available. The data need to be processed before being consumed by applications. Context management is essential for IoT applications, it collects, transforms, and aggregates the data. Components of an IoT system such as IoT platforms and IoT middleware, which have the role of abstracting communication between IoT applications and IoT devices, are crucial when it comes to context management.

2.2 IoT Middleware Definition and Architecture

Middleware can be defined as a *"software that resides between applications, services, and their underlying distributed architecture and platforms"* [Blai16] and *a distributed software layer that sits above the network operating system and below the application layer and abstracts the heterogeneity of the underlying environment.* [Mahm04]. Therefore, the role of middleware is to manage the complexity and heterogeneity of distributed infrastructures providing a straightforward programming environment for distributed-application developers [Camp99]. With the growth of distributed devices in the IoT, the role of middleware is even more important to manage the massive data consumed by applications.

A remarkable characteristic of the IoT paradigm is the high heterogeneity in terms of hardware and software, encompassing devices with different capabilities, functionalities, and network protocols. To tackle such heterogeneity, *IoT middleware* have been proposed in the last years towards abstracting away specificities of devices from applications and users, promoting interoperability, and providing services and interfaces to leverage application development [Blai16, Lee15, Ngu17].

There are several definitions for IoT middleware such as *"a software system designed to be the intermediary between IoT devices and applications"* [Ngu17] and *"the software technology that has been used as the basis for the development, management, and integration of both heterogeneous devices and applications in IoT environments"* [Amar16]. Following these definitions and the common characteristics of IoT middleware [Marq17, Razz16] we can define it as a software layer residing between the application and physical sensing layers (see Fig. 2.2) that provides abstractions for accessing physical devices (Physical Sensing Layer) and managing data produced by them, thereby enabling developers to focus on the application development itself (Application Layer). IoT middleware mainly provides (i) support to the interactions between devices and users, it handles interoperability for managing hardware and software entities (IoT Middleware Layer); (ii) virtualization of devices (Device Discovery); (iii) data collection and aggregation (Context Management).

IoT middleware is a glue layer in the IoT architecture that will enable communication between IoT devices and IoT applications, abstracting and moving the complexity from the application to the middleware.

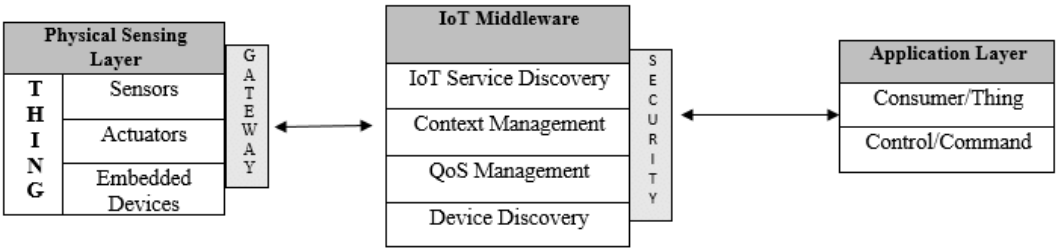


Figure 2.2. IoT system basic architecture [Onor18]

2.3 IoT Platform Definition and Architecture

Asghari and co-authors [Asgh18] mention that a new generation of applications such as smart buildings, smart cities, agriculture, logistics, and supply chain manufacturing, all share common requirements such as interacting with IoT devices according to various IoT protocols. In addition, applications should all offer extra-functional requirements such as security, fast response time, low energy consumption, availability, reliability, and high throughput. In that context, the usage of IoT platforms to support IoT application deployment is a recent trend: IoT platforms provide services to deploy and run applications on top of a hardware and/or software suite [Nakh15]. IoT platforms have been proposed to deal with the high degree of hardware/software heterogeneity in IoT environments, providing the necessary services for IoT applications to be built on top of them. They may be deployed and shared by many applications.

There are several definitions spread through the published research about IoT platforms: "*It is middleware and the infrastructure that enable the end-users to interact with smart objects*" [Mine16]. This first definition introduces the IoT platform as a way to connect to IoT devices (or smart objects) by providing an infrastructure where IoT applications can be built on top. "*It is software that enables communication with remote devices through the Internet, and provides services such as programming frameworks, M2M integration, data, and device management, security and storage*" [Sing17]; The second is more detailed about the features an IoT platform needs to have to be considered as a platform. "*It is software that connects the edge of devices, gateways, and data networks to cloud services and applications*" [Heja18]. Finally, the third definition shows the central position of the platform between the distributed components of an IoT system.

2.4 Analysis of common features of three context management IoT Platforms

For this work, we consider context management IoT platforms, which constitute a subset of IoT platforms that provide middleware components for accessing IoT data. Context management platforms provide virtualization and abstraction of the sensors (a virtual representation of a sensor in software), the discovery of sensors, and the provision of context data to the applications.

We have selected three context management IoT platforms, two of them are widespread platforms used by the community (i.e. *Orion* and *OM2M*), the third one is a research platform (muDEBS). For those three platforms, we have studied their common features. The objective of this study was to determine which abstractions should be provided by an IoT middleware that interacts with those platforms. We present the three selected IoT platforms in Section 2.4.1, then we study the common abstractions provided by those three platforms: context data models in Section 2.4.2, interaction patterns, APIs and protocols in Section 2.4.3, then we present their discovering and filtering capabilities in Section 2.4.4. Finally, we present a synthesis of this study in Section 2.4.5.

2.4.1 Description of the selected IoT platforms

In this section we briefly present the ecosystem and main characteristics of three context management IoT platforms: (i) FIWARE comes from the European community, (ii) OM2M is an M2M standard, and (iii) muDEBS is a research platform focusing on scalability for heterogeneous systems. These platforms were selected based on the services they provide to enable IoT applications to be either built on top of them or for a part of the complexity of the communication with IoT devices to be abstracted by the platform.

FIWARE is an IoT platform supported by the European Community. It is a generic, open-source solution, and it provides many extensible, reusable and interoperable components to enable easy system development in different application domains, the so-called *Generic Enablers (GEs)*. FIWARE encompasses GEs for purposes such as context entity management, device management, historical data storage, event processing, security, and the creation of dashboards. The main FIWARE GE is the Orion Context Broker (or simply Orion), which is the context management IoT platform.

The OM2M platform is an open-source implementation of the oneM2M

standard [oneM]. It provides a Machine-2-Machine service that offers developers the capability to develop services independently of the underlying network. Furthermore, it is built as a modular architecture. oneM2M provides a horizontal Service Common Entity (CSE) that can be deployed in an M2M server, a gateway, or a device. Each CSE provides application enablement, security, triggering, notification, persistence, device inter-working, and device management. The platform exposes an API providing many other services such as resource discovery, application registration, and context data management.

muDEBS stands for *Multiscale Distributed Event-Based System* and it is a research IoT platform designed for allowing the dissemination of data in large-scale and heterogeneous systems involving clouds, cloudlets, desktops, laptops, mobile phones, and smart objects of the Internet of Things. The muDEBS platform introduces multi-scoping for limiting the broadcasting of subscription filters and enabling to forward notifications only to relevant scopes of the overlay network of brokers. The platform brings into play producer and consumer contracts. Broker nodes form an overlay that supports cycles and delimits scopes to implement localized scalability—i.e., filters and data are tagged with scoping meta-data that are specified in the producer and consumer contracts, and they are then broadcast only to relevant scopes, not to the whole system.

2.4.2 Context data-model

The IoT domain is characterized by a high degree of heterogeneity because of the many available technologies [Li15]. Devices from different manufacturers expose their data following various data models and using different protocols. One challenge for the IoT platform is to provide a unified data model. The model must include location data, e.g. the current location of the device, and quality attributes (e.g., freshness, resolution). Furthermore, a well-defined data model facilitates the development of wrappers and adaptors to parse the data in IoT applications. We present below the context data models used by the three studied IoT platforms.

Orion data model

Orion follows the Next Generation Service Interface (NGSI) data model [NGSI] to standardize information exchange and to allow for interoperability among components. In NGSI, the information is structured generically through entities that can be used to represent both physical and virtual elements such as a

building, a car, sensors, and actuators. As depicted in Figure 2.3, an NGSI Entity has an identifier, a type, and a list of attributes. Attributes have a name, a type, a value, and a list of metadata, which have the same data structure as the attributes.

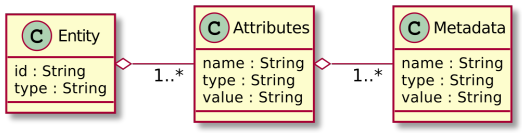


Figure 2.3. Next Generation Service Interface (NGSI) entity model

The generic model is serialized in JSON. Listing 2.1 shows the representation of an entity and its attributes. The entity is a sensor near the Eiffel Tower; the attributes are the recorded temperature (Line 4), the location (Line 14), and the temperature resolution (Line 18). All the attributes are specified as additional properties in JSON and support metadata declaration. Furthermore, Orion's data model provides support for localized data: When entities are registered through the API, they can send their geographical coordinates as an attribute.

Listing 2.1. "Representation of temperature sensor data in Orion"

```
1  {
2    "id": "temperature_eiffel_tower_310",
3    "type": "LM35",
4    "temperature": {
5      "value": 23.3,
6      "type": "Float",
7      "metadata": {
8          "Unit": {
9              "value": "Celsius",
10             "type": "String"
11         }
12     }
13   },
14   "location": {
15     "value": "48.6223426, 2.4404356",
16     "type": "geo:point"
17   },
18   "temperature_resolution": {
19     "value": "0.1",
20     "type": "Float"
21   }
22 }
```

An important point to highlight is the capability to model quality attributes. This information can be represented as attributes of the entity or as metadata. temperature_resolution is a quality attribute (Line 18) and is represented in the example as an attribute, but Celsius (Line 9) is represented

as metadata. This gives more flexibility to the developer to virtualize the sensor/device.

OM2M data model

The OM2M IoT platform conforms to the data model of the oneM2M specification. Figure 2.4 shows the generic model used in the OM2M platform. The model contains ResourceType that includes child resource types and specifically named resource attributes, with each specific attribute having its value.

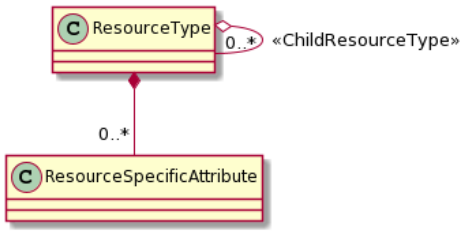


Figure 2.4. OM2M generic data model.

Listing 2.2 shows the data model being used in a concrete example to represent a sensor resource inside the platform. The resource contains named attributes such as creation time (ct, Line 7), resource name (rn, Line 3), and child resources (ch, Line 18)) that are linked to the values that the sensor is sending to the platform. The sensor named temperature_eiffel_tower_310 gives readings about its temperature (Line 19), location (Line 23) and temperature resolution (Line 27). These values are accessible via the link in attributes named "val" (Lines 21, 25, and 29).

Listing 2.2. "Representation of temperature sensor data in oneM2M"

```
{
   "m2m:ae" : {
      "rn" : "temperature_eiffel_tower_310",
      "ty" : 2,
      "ri" : "/in-cse/CAE413477219",
      "pi" : "/in-cse",
      "ct" : "20200520T163516",
      "lt" : "20200520T163516",
      "lbl" : [
                        "Category/temperature",
                        "Location/home",
                        "Type/sensor"
                    ],
      "acpi" : [ "/in-cse/acp-385742697" ],
      "et" : "20210520T163516",
      "api" : "app-sensor",
      "aei" : "CAE413477219",
      "ch" : [ {
         "nm" : "temperature",
         "typ" : 3,
         "val" : "/in-cse/cnt-432974030"
      }, {
         "nm" : "location",
         "typ" : 3,
         "val" : "/in-cse/cnt-490739007"
      }, {
         "nm" : "temperature_resolution",
         "typ" : 3,
         "val" : "/in-cse/cnt-970640701"
      } ],
      "rr" : false
   }
}
```

There is no limit to what data can be included in the data model. For instance, quality attributes and location data can also be included as Resource Specific Attributes.

muDEBS data model

The muDEBS platform uses a semi-structured data model “à la” XML and then allows XPATH expressions in scripts written in the JavaScript language. Devices and clients exchange any kind of data. Clients publish contracts to the brokers connected through the IoT platform and any device that has data that is interesting for the IoT application is forwarded to applications by brokers: data filters are expressed in JavaScript. Figure 2.5 shows the data model used inside muDEBS to represent entities. A context report aggregates a set of context observations, with a context observable defining what is observed, e.g. a temperature sensor. The observable is linked to a context entity that can

have a relation with other entities. Furthermore, the model also contains an optional MultiscalabilityReport including the scope (e.g., geographically, administratively) of the report that may be used by muDEBS for filtering purposes.

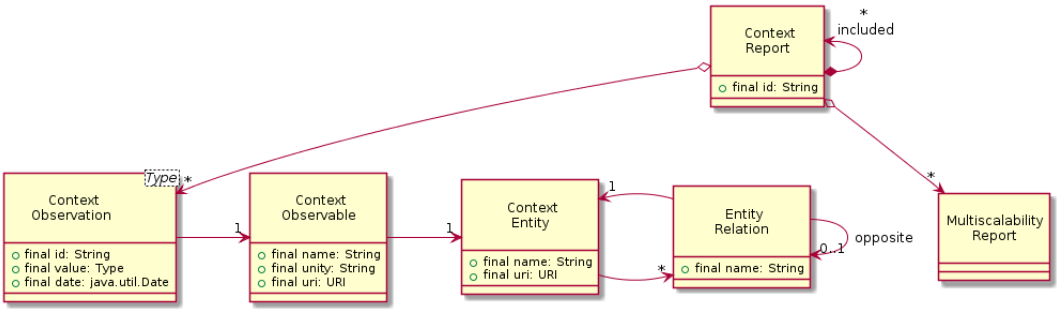


Figure 2.5. muDEBS generic data model.

Listing 2.3 shows what an entity that represents a temperature sensor in the Eiffel Tower looks like when going from the metamodel to a concrete model. There is a ContextReport that contains attributes (id, value and date) and observables. The attributes are characteristics of the entity that can be looked up by consumers while observables are values that can be observed by client applications to receive updates over these values.

Listing 2.3. "Representation of temperature sensor data in muDEBS"

```
<ContextReport>
    <Observation>
        <id>temperature</id>
        <value>
            <temperature>23.3</temperature>
            <resolution>0.1</resolution>
        </value>
        <date>22-02-2022-15:13:55</date>
        <observable>
            <name>temperature_eiffel_tower_310</name>
            <unity>Celcius</unity>
            <uri>http://192.168.0.1/temperature</uri>
            <ContextEntity>
                <name>Eiffel Tower</name>
                <uri>http://192.168.0.1</uri>
            </ContextEntity>
        </observable>
    </Observation>
</ContextReport>
```

The model offered by the muDEBS platform also makes it possible to provide localized data and quality attributes. When the contract sent by the client contains the need for a device to send the location and a given quality of the attributes, brokers may filter out context reports that do not contain these

data.

Synthesis

The three presented data models allow IoT system designers to define metadata for the sensors (e.g, location of the sensor, unity, resolution) as well as for the sensed values (e.g. freshness). However, the meta-models used by the platforms are different. As a consequence, applications should be able to unmarshal collected data differently for each IoT platform.

2.4.3 Interaction patterns, APIs and protocols

In this section, we present the interaction patterns, APIs, and protocols provided by the three studied IoT platforms. Two interaction patterns are commonly provided by IoT platforms: synchronous requests and publish/subscribe. In the synchronous pattern, a client application consumes IoT data by sending a request to the IoT platform, which responds to the request. In the publish/-subscribe pattern, a consumer application registers to the IoT platform and asynchronously receives publications when they are available [Reev13].

All the IoT platforms propose one or several APIs, i.e. a set of functions that may be called by client applications, and protocols used for interactions between IoT platforms and IoT consumer applications.

Interactions with Orion

In FIWARE, Orion provides both synchronous and publish/subscribe interaction patterns. The platform provides REST APIs for interactions with context producers and IoT application consumers [FIWA]. HTTP is used both for synchronous and publish/subscribe interactions. When using the synchronous pattern, the application developers make HTTP calls to the API following a request/response behavior. On the other hand, for the publish/subscribe interaction pattern, the consumer application uses the HTTP API to subscribe to content and waits for HTTP requests from the broker to receive data.

Interactions with OM2M

The OM2M platform provides the two interaction patterns. The synchronous pattern uses a REST API: The consumer application makes HTTP requests and receives data [OM2M]. The publish/subscribe pattern uses the MQTT protocol [OASI15]: A consumer application subscribes to topics using the MQTT

protocol, and then the OM2M platform sends publications to the subscribed consumer applications. As a consequence, the platform benefits from MQTT characteristics such as different qualities of services (from "at least once" to "exactly once"), clean sessions, and retaining flags for managing disconnections.

Interactions with muDEBS

The muDEBS platform provides the publish/subscribe interaction pattern through an API that is *ad hoc*. The synchronous mode is emulated by providing a consumer application with the data that have last been produced by producers: These data are returned to the consumer when the subscription filter of the consumer is installed onto access brokers; such brokers manage direct connections to producers.

Synthesis

The APIs offered by the 3 platforms are different and they support interaction patterns implemented through different IoT protocols. Applications have to adapt themselves to each platform's interaction capabilities.

2.4.4 Discovery facilities and filtering capabilities

IoT platforms include services to find context producers and also to filter the data from these producers. Filtering data ensures better control over the quality and the amount of data received by applications, avoiding unnecessary interactions with the IoT platform. In this section, we present the facilities provided by the three IoT platforms.

Discovering and filtering with Orion

Using FIWARE, via the Orion Context Broker, application developers make HTTP requests to the API by providing parameters such as identifier, type, and attributes, to select entities. The filtering capability uses a Simple Query Language, which is easy to implement and use (e.g. `q=temperature>40; humidity>40`), and also Geographical Queries that allow to filter by geographical location (e.g. all the entities located closer than 5 km from a point is expressed by `georel=near; minDistance:5000 & geometry=point & coords=-40.4,-3.5`).

Discovering and filtering with OM2M

The discovery functionality in OM2M is implemented using a HTTP GET request passing parameters such as `fu`, which stands for "filter usage", and `lbl`, which stands for "label". For instance, to get a temperature sensor, the application would use `fu=1 & lbl=Type/sensor`, where `fu=1` indicates that it is a discovery request. Furthermore, the discovery functionality enables retrieval of the set of resource URIs matching a specific filter criterion. The OM2M platform is not able to filter data that is sent by the sensors, leaving a gap for unneeded data to be sent to the application. Although, when using the publish/subscribe pattern, the application can specify the maximum frequency of notifications.

Discovering and filtering with muDEBS

muDEBS does not need any discovery functionality because context reports describe observable entities and filters are content-based, i.e. subscription filters can select the entities from which data are to be received (e.g. temperature greater than 20 degrees Celsius at the Eiffel Tower). In addition, muDEBS filtering functionality is extended into (producer and consumer) contracts to include quality of data [Mari14] and privacy concerns [Deni20]: e.g. good precision of air quality data only, or only authorized end-users. Brokers make sure that only relevant data are forwarded to consumers.

Synthesis

Filtering capabilities of the three studied platforms vary: content-based filtering is possible with muDEBS and Orion while OM2M is able to limit the frequency of the notifications.

2.4.5 Synthesis of the context management IoT platforms analysis

Table 2.1 summarizes important features that the FIWARE, OM2M and muDEBS platforms bring into play. These features are: (i) **Interaction Pattern** that are separated into synchronous or publish/subscribe patterns; (ii) **Application Protocols**, showing how many and what are the protocols supported by the platform for applications to communicate with; (iii) **Data Models**, the standards used by the platforms to send/receive data from/to applications; (iv) **Discovery Service**, the process of automatically finding

appropriate services and their providers by taking into consideration the context and QoS (Quality of Service) of requests; and (v) **Filtering Capabilities**, indicating how the platform can refine data sent/received from sensors, devices, and applications.

In summary, in this section, we have shown that the FIWARE, OM2M, and muDEBS platforms provide different data models. As a consequence, an IoT middleware should provide specific data unmarshallers for each IoT platform. Furthermore, the IoT platforms have each their way of communicating. The Orion platform relies on HTTP to do synchronous and Pub/Sub communication. The OM2M platform relies on HTTP for synchronous communication and MQTT for Pub/Sub communication. The muDEBS platform uses its protocol based on the AMQP specification. With each IoT platform having its communication protocol, the effort to develop IoT applications increases as it is more time-consuming to understand and implement each protocol. An abstraction on top of these different IoT platforms could be provided by an IoT middleware in order to lower the effort necessary for the developer to implement the interactions with IoT platforms. In this Ph.D. we have designed such a middleware that we present in Chapter 4.

Table 2.1. Comparison table

	Platforms		
Characteristics	FIWARE/Orion	OM2M	muDEBS
Interaction Patterns	Sync / PubSub	Sync / PubSub	PubSub(/Sync)
Application Protocols	HTTP / HTTP	HTTP / MQTT	*ad hoc*
Data Models	NGSI	oneM2M	*ad hoc*, semi-structured
Discovery Service	√	√	√
Filtering Capabilities	√	×	√

2.5 Measuring software energy consumption

Nowadays, there is a challenge for measuring the energy consumed by software. One objective to use energy measurement tools is to increase the knowledge of where in the application there is a need for reducing energy consumption and drive better development towards energy-efficient/aware strategies.

More particularly, in this study, we aim to measure and reduce the energy consumption of IoT consumer applications. As those applications deal with a considerable amount of received data, a part of their energy consumption

comes from their communications. Additionally, any software used by the application, and especially the IoT middleware, impact the energy consumption of the application. One of our objectives will be to identify and reduce the energy consumption of IoT middleware themselves.

The energy consumption could be measured with different tools. A classical tool to measure energy consumption in devices is the Watmetter, but there are also several software tools such as *RAPL*, *Likwid*, *PowerAPI* and *JoularJX*. Each of those tools provides different levels of detail from the total consumption of a computer to a fine analysis of the consumption of portions of codes (e.g., analysis between two lines of code or methods). In this section, we study some of those tools with their particularities, aiming to better choose the tool to be used in our contributions.

2.5.1 Wattmeter

Wattmeters are hardware devices widely used to measure the energy of any type of device. A typical wattmeter can measure voltage (V), current (A), power (W), power factor (pf), and energy consumption (kWh). The measurement accuracy usually varies within the range of $\pm 1\%$ to $\pm 5\%$ depending on the brand and model of the device. As an example, a wattmeter with a measurement accuracy of $\pm 2\%$ can show the power of 100 W between 98 W or 102 W. There are three basic types of wattmeters, (i) Electrodynamometer, (ii) Induction, and (iii) Digital, each with their characteristics.

Electrodynamometer

The Electrodynamometer wattmeter has two coils i.e., fixed and moving coils. The fixed coil is connected in series with the circuit to measure power consumption. The supply voltage is applied to the moving coil. Current across the moving coil is controlled with the help of a resistor, which is connected in series with it. The moving coil on which the pointer is fixed is placed in between fixed coils. Two magnetic fields are generated due to the current and voltage in the fixed coil and moving coil. The pointer deflects as the two magnetic fields interact. The deflection is proportional to the power that is flowing through it. Figure 2.6 shows the basic circuit diagram of the electrodynamometer, this design is simple, reliable, and rugged. The wattmeter takes a current I from a power supply and is separated into two other currents, I_c for the current coil, which links to where the energy "LOAD" is used, and I_p which is the pressure coil linked to a resistance.

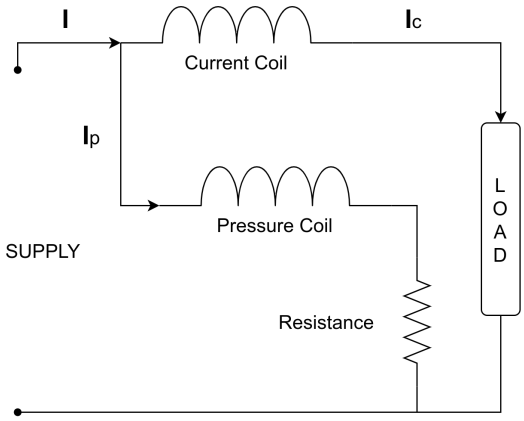


Figure 2.6. Electrodynamometer

Induction

The induction wattmeter, shown in Figure 2.7, consists of two electromagnets with a laminated core made of steel silicon: shunt and series magnets. Between these electromagnets, there is a thin disk of aluminum, the aluminum disk receives a variable magnetic field and an induced current appears on the disk. The induced current in the presence of a magnetic field produces a torque on the disk, that is connected to a pointer. The copper rings on the shunt magnet have adjusted position so that the generated flux and supply voltage has a phase difference of 90 degrees. This type of wattmeter can only be used when the frequency, voltage, and temperature are constant. However, this wattmeter can only operate at the frequency that was designed. On the other hand, it does not suffer the effects of external magnetic fields.

Digital

Digital wattmeters measure current and voltage electronically thousands of times a second (depending on the supplier), multiplying the results in a microcontroller to determine watts. The controller can also perform statistics such as peak, average, low watts, and kilowatt-hours consumed depending on the brand used. Furthermore, with the enhanced capabilities of having a microcontroller, they can monitor the power line for voltage surges, and outages, and even communicate with external systems. Figure 2.8 shows a basic diagram of how a digital wattmeter can be structured by receiving voltage and current from the device that needs to be measured and passing this through a microcontroller that will output energy data such as energy consumption.

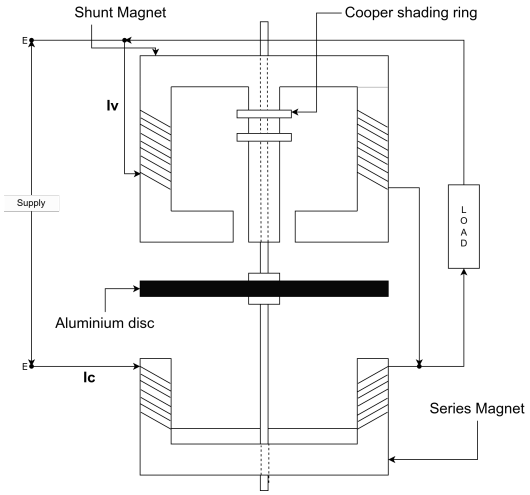


Figure 2.7. Induction wattmeter

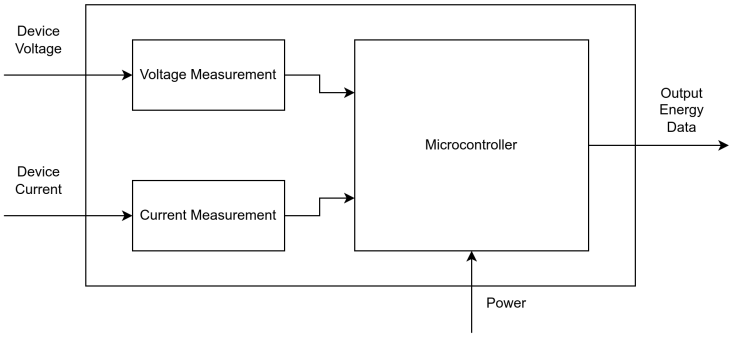


Figure 2.8. Digital wattmeter

Wattmeter Usage

The choice of using a type of wattmeter for energy measures can be dependent if the type of the current is alternating current (AC) or direct current (DC). The electrodynamometer is an AC and DC power measurer that is better used with high power factors and where there is no external magnetic field to impact the results, ensuring high accuracy and uniform scale. Differently, the induction wattmeter can only be used for AC power and is unaffected by stray magnetic fields, but high temperatures can cause errors and have lower accuracy. Finally, the digital wattmeter works on a different principle that uses a microprocessor to sample the voltage and current thousands of times a second for either AC or DC. For each sample, the voltage is multiplied by the current at the same instant. Digital wattmeters vary considerably in correctly calculating energy

as its dependent on the brand and how it is constructed.

Concerning IoT consumer applications, the choice of using the wattmeter nowadays is mainly to use a digital wattmeter. There is a high amount of different brands of wattmeters and the choice will be mainly over the characteristics such as refresh rate, maximum current, working voltage, and accuracy. Moreover, there could be also a consideration of how these measures can be retrieved. Some brands of wattmeter provide APIs that can be used to read the energy data, which could ease the development of code to analyze the energy consumption of IoT consumer applications.

2.5.2 RAPL

The thermal design power (TDP) represents the maximum amount of power (heat) the cooling system in a computer is required to dissipate. For example, for an Intel processor with a TDP of 35W, Intel guarantees the Original Equipment Manufacturer (OEM) that if it implements a chassis and cooling system capable of dissipating that much heat, the chip will operate as intended. However, processors can operate over the TDP limit and consume more energy for a short amount of time, but it will also increase the amount of heat generated. This won't necessarily violate the TDP, but the processor needs to have more heat dissipated to keep the processor running at the speed it was designed for as too much heat can potentially break it.

To overcome heating problems, the Power Control Unit (PCU) inside the CPU uses internal models and counters to predict the actual and estimated power consumption. The Sandy Bridge microarchitecture from Intel added onboard power meter capability that exports power meters and power limits used in its calculations through MSRs (Machine Specific Registers). This interface is called Running Average Power Limit (RAPL).

RAPL provides a way to set power limits on processor packages and DRAM. It enables monitoring, controls programs, and dynamically limits max average power, matching the expected power and cooling limits for the processor. RAPL can do this using a set of counters that provide energy and power consumption information. It is not an analog power meter, as it uses a software power model. This model estimates energy usage by using hardware performance counters and I/O models which are accurate to actual power measurements [Rote12].

Although RAPL uses information from the CPU and can accurately measure its consumption, the energy consumption information is still limited to the amount of processing passed to the cores of the processor. Other energy

consumption information from sources such as network cards and disks is not accounted for without an external tool. For measuring the energy consumption of network heavy applications, such as IoT consumer applications, the network energy consumption could be an imperative indicator to see better what is the impact of the application on the energy consumption.

2.5.3 LIKWID

LIKWID ("Like I Knew What I'm Doing") [Trei10] is a set of easy-to-use command line tools to support optimization. It is targeted toward performance-oriented programming in a Linux environment. It does not require any kernel patching and is suitable for Intel and AMD processor architectures. In the case of Intel processors, LIKWID is capable of reading the RAPL information and provides a tool called "likwid-powermeter" which allows you to query the energy consumed within a package for a given period and computes the resulting power consumption among other resources such as TDP, active cores, the temperature of CPU cores, etc.

LIKWID also offers instrumentation support called MarkerAPI, for C, C++, and Lua. This instrumentation consists of function calls and also enables the measuring of code regions to get a better insight into the system's activities executing the code. A developer that uses the MarkerAPI will be able to declare markers inside the code and LIKWID will be able to measure CPU time usage. Moreover, the MarkersAPI works with the "likwid-perfctr" tool which will enable the developer to have much information about the CPU usage including energy.

It is important to highlight that LIKWID measures the resource usage of the whole processor (core) and for accurate measures, there is a need to make sure that the process being measured is running on one or more cores that LIKWID is currently gathering information from. This is especially important for multi-threaded processes as there is a need to bind the process to a fixed number of cores to ensure both the proper functioning of the process and LIKWID. This could become an issue when dealing with multithreaded applications which would imply the usage of available cores by the Operational System (OS) unless there is an external way to modify this behavior.

2.5.4 Power API

PowerAPI [Bour13, Nour12] is an application programming interface (API) to monitor the energy consumption of applications at the granularity of system

processes. The API uses power models for estimating the energy consumption of processes and software blocks of code. The models and estimations are divided by hardware resources and by software applications. Concretely, PowerAPI can estimate the energy consumption of a running process for the CPU, the hard disk, or both or more hardware resources.

The architecture of Power API is shown in Figure 2.9. The architecture is modular and contains some modules such as the power module that can be started and stopped at runtime whenever needed. A set of sensor modules (e.g., CPU, network) collect raw information about hardware resource utilization, either directly from the devices or through the operating system. This information is then exposed to another set of independent formula modules that use the power models and provide an estimation of the energy consumption of the computer resources.

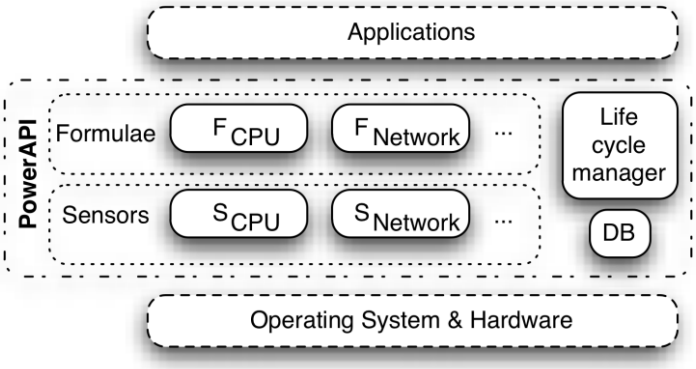


Figure 2.9. Power API architecture [Nour12]

The usage of Power API to measure energy consumption will be detailed across hardware such as CPU, Disk, and RAM. However, the details are still linked to the process itself leaving a gap in more specific energy measures from pieces of the code. Detailed energy data over code could lead to a better view of what parts of the software running are using more energy and show the developer where to focus when there is a problem with energy constraints.

2.5.5 JourlarJX

JoularJX [Nour22] is a Java-based agent for power monitoring at the source code level with support for modern Java versions and multi-OS to monitor the power consumption of hardware and software. Using JoularJX, an application developer simply hook it to the Java Virtual Machine when starting the

Java program being tested. To get power readings, JoularJX uses a custom "PowerMonitor" [1] program on Windows, and "Intel RAPL" on GNU/Linux. Furthermore, it supports Raspberry Pi devices (multiple models) running with GNU/Linux and the energy consumption calculations are estimated using a research-based regression model provided by the developers of the tool.

When an application is running using JourlaJX as a Java Agent, the measures are retrieved every 10ms by reading the stacktrace of the Java code, then CSV files are generated during runtime with real-time power data of each method and are overwritten each second with the new power data. When the program exits, JoularJX generates two new CSV files with the total energy of all monitored methods. The first file contains power or energy data for all methods in the code, including internal java methods. The second contains a filtered file that only includes the power or energy data of selected methods.

Having detailed energy data about the methods in the code is a strategy to introduce more information about the general health of software, especially with high energy constraints. JoularJX can give this detailed energy information, but it is limited to the processor and DRAM energy consumption. Depending on the software, other hardware components of the computer could be used a lot (Disk, network card, etc). This extra information on energy consumption could point to other pieces of code that need a change to fulfill possible energy constraints the software might have.

Considering IoT consumer applications, the usage of JoularJX can bring many advantages given that the application itself is developed using Java. The information on the methods of the code in detail can pinpoint the energy consumption issues that the application has and enable easier implementation of energy-efficient/aware strategies.

2.5.6 Synthesis

Measuring the energy consumption of software is possible as hardware evolves to integrate sensors and models. Software tools for energy measurement are also improving to give more precise information. This scenario enables the development of frameworks to be able to save energy by knowing how much energy is consumed by software. Table 2.2 summarises the main differences among the different tools presented (Wattmeter, RAPL, LIKWID, Power API, and JourlarJX). These are the levels of measure that can be computer, CPU, process, or code, and also how the calculation is done.

[1] https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html

Table 2.2. Comparison of energy measurement tools

Tool	Computer level measure	Processor+DRAM level measure	Process level measure	Code level measure	Calculation
Wattmeter	√	×	×	×	External Hardware
RAPL	×	√	×	×	Sensor + Power Model
LIKWID	×	√	√	√	RAPL
Power API	×	√	√	×	Power Models
JourlarJX	×	×	√	√	RAPL/Power Models

The tools for energy measurement gathered for the presented comparison show that when dealing with software tools, the tools built upon RAPL (i.e. JoularJX and LIKWID) are focused on the CPU and DRAM consumption. Two tools (i.e. JoularJX and LIKWID) deal with code-level energy consumption. Furthermore, the wattmeter is usually used for computer-level measurement (it could also be capable of CPU-level measures, but this would involve complex hardware modifications in the CPU to allow the connection with a wattmeter). Also, although it measures the whole computer energy consumption, the data of the energy consumed is very close to the real consumption, whereas the other tools are based on estimations through power models. Finally, using a wattmeter, it is possible to measure the consumption of any computer with any processor, but using RAPL, only Intel processors can be used. LIKIWID, Power API, and JourlarJX support different types such as Intel, AMD, and ARM. The support for multiple CPUs is an essential requirement for software energy measuring tools to foment the need for further energy efficiency in hardware and software.

2.6 Conclusions

The IoT paradigm is a popular domain that encompasses multiple smaller domains including IoT platforms, middleware, and protocols. The IoT applications that are being developed need interaction with many heterogeneous components that not only present different IoT protocols but also different communication patterns such as publish/subscribe and request/reply. To overcome this, IoT systems provide software and infrastructure such as IoT platforms to hide IoT complexities, lowering the effort necessary to build IoT applications.

However, the lack of standardization has led each IoT platform to propose its abstractions, APIs, and data models. As a consequence, programming interactions between an IoT consumer application and an IoT platform are time-consuming, error-prone, and depend on the developers' level of knowledge about the IoT platform. In addition, application developers should also tame energy consumption constraints. For this concern, tools for energy measurement like wattmeters and software measurement tools provide a set of features that can be an interesting complement to mastering energy consumption and learning the best strategies for interacting with IoT platforms.

Chapter 3

IoT Middleware and Energy-Efficiency Related Work

This chapter presents an analysis of related work concerning IoT middleware and energy-efficiency/awareness in IoT middleware and IoT protocols. Section 3.1 presents the main findings over an extensive search using a systematic literature review (SLR) method about energy-efficiency/awarenss in IoT middleware. Although, with this study we were able to present findings over the current literature on energy-efficiency/awarenss in IoT middleware, other more general works regarding IoT platforms and IoT middleware in general are not included. Section 3.2 shows an analysis of papers that present evaluations over HTTP and MQTT protocols showing the energy consumption of these protocols under different conditions. Section 3.3 shows different types of IoT middleware for consumer applications and how they are able to abstract the complexity of interactions with other IoT services through different strategies. Finally, Section 3.4 gives the summary and final remarks of the chapter.

3.1 Literature Review on energy-efficiency in IoT middleware

The literature presents many studies related to energy-efficiency/awareness in areas such as Big Data [Wei14, Wu16], communication technologies [Bian12, Chao11, Mogh15, Wu18], industry [Shen15, Wang16], and general energy-saving techniques [Este15, Shai17a]. However, as far as it is concerned, there is still no study offering an overview of the state of the art on how IoT middleware platforms can contribute to energy efficiency and energy-awareness in IoT systems. Such an overview can enable researchers and practitioners to critically reflect on the current state of the art, identify critical issues to drive future research on energy requirements in IoT systems. To fill this gap, a systematic literature review (SLR) on energy-awareness and energy efficiency in IoT middleware was done. The review was carried out by following a systematic, rigorous procedure driven by well-established guidelines for collecting, selecting, and analyzing primary studies while proving scientific value to the obtained findings [Kitc11, Pete08].

The systematic literature review on energy-efficiency/awareness aimed to investigate: (i) what is currently provided at the middleware level to better manage the energy consumption of IoT systems; (ii) which abstractions exist at the IoT middleware level to enable energy-awareness; (iii) how the existing proposals have been evaluated and their maturity; and (iv) how to deploy IoT middleware with different strategies for limiting energy consumption and enabling energy efficiency in IoT systems.

To present the literature review in this chapter, the sections are structured as follows. Section 3.1.1 presents the research methodology adopted in this work in terms of the research questions to be answered and study search selection strategies, as well as details on how the relevant primary studies were selected. Section 3.1.2 presents a synthesis resulting from the analysis of the selected primary studies as answers to the research questions. Finally, Section 3.1.3 summarizes the main findings of this work along with some concluding remarks.

3.1.1 Research methodology

The SLR was carried out considering well-defined, consolidated guidelines available at the literature [Kitc16, Pete08]. Three basic steps were followed: (i) *planning*, which gives rise to a protocol defining the research questions to an-

swer, the search strategy to be adopted, the criteria to be used to select primary studies, and the data extraction and synthesis methods; (ii) *execution*, in which studies are identified, selected, and analyzed according to the protocol; and (iii) *reporting*, which aggregates information extracted from relevant studies considering the research questions and outlines conclusions based on them.

Research questions. Four research questions (RQs) were proposed aiming at finding primary studies to understand and summarize pieces of evidence about energy-efficiency and energy-awareness in IoT middleware:

RQ1: How energy-efficiency has been addressed by IoT middleware?
Rationale: To identify proposals that consider energy efficiency in IoT middleware.

RQ2: What abstractions are provided by middleware for enabling energy-awareness or energy efficiency of IoT systems?
Rationale: To identify any synergy between IoT middleware and applications atop them either for energy-awareness or energy-efficiency transparently provided by middleware.

RQ3: How have existing energy efficiency proposals been evaluated?
Rationale: To identify how to measure or evaluate energy consumption within the many components of an IoT system.

RQ4: Which part/component of the distributed architecture is considered in the middleware solution for energy efficiency?
Rationale: To identify where the IoT middleware is deployed, what is measured, and for which component of an IoT system the energy consumption is reduced.

Answering RQ1 aims to result in a catalog of strategies and techniques that have been used so far in IoT middleware for energy efficiency. Answering RQ2 gathers data about how the research community deals with the software level when considering middleware and energy-awareness and energy efficiency. Answering RQ3 gathers knowledge on how the energy efficiency of IoT middleware and systems built with their support have been evaluated. Answering RQ4 aims to provide information on where energy-related measurements are made and which components are affected by the energy efficiency techniques adopted by IoT middleware platforms.

Search strategy. An automated search process was carried out over five electronic publication databases to retrieve relevant studies to answer the posed

RQs. The used databases were: IEEEXplore[1], ACM Digital Library[2], Scopus[3], Science Direct[4], and Web of Science[5]. These sources are among the most popular publication databases in Computer Science and Engineering and have good coverage of the literature [Dybå07, Kitc16]. Other relevant criteria were the quality of the results returned by the automated search procedure, the availability of the full text of the studies, ease of use, up-to-dateness of contents, and versatility to export results.

Based on the defined RQs, three main terms were initially identified, namely *Internet of Things*, *middleware*, *energy efficiency*, and *energy-awareness*. The following search string was built considering synonyms and alternative terms:

```
middleware AND (energy consumption OR energy efficiency OR energy
      awareness OR green) AND (Internet of Things OR IoT)
```

Selection criteria. Selection criteria were used to assess the relevance of each primary study to answer the established RQs. Inclusion criteria define circumstances that make a study relevant, whereas exclusion criteria exclude studies that are unrelated to the RQs. Four inclusion criteria (ICs) and five exclusion criteria (ECs) were defined. In this work, a given primary study was regarded as relevant if it did not meet any EC and met at least one IC.

IC1: The study addresses energy efficiency or energy-awareness in the IoT context.
IC2: The study presents an energy efficiency proposal relying on middleware or middleware models.
IC3: The study presents a proposal to reduce energy consumption in IoT applications or middleware.
IC4: The study concerns software that connects systems to reduce energy consumption.

EC1: The study is not related to IoT or middleware platforms in this context.
EC2: The study is a previous version of a more recent study on the same research.
EC3: The study does not have an abstract, or the full text is unavailable.
EC4: The study is a table of contents, foreword, tutorial, editorial, keynote talk, or summary of conference/workshop.

[1] http://ieeexplore.ieee.org
[2] http://dl.acm.org
[3] http://www.scopus.com
[4] https://www.sciencedirect.com/
[5] https://www.webofknowledge.com/

EC5: The study is not written in English, the most common language in scientific papers.

Data extraction. A data extraction sheet was built with items related to the RQs and other relevant information to extract data from the selected primary studies. Besides basic information such as title, publication year, and venue, extracted data concerned (i) the proposal of the presented middleware, (ii) where the middleware is deployed, (iii) the impact of the energy efficiency strategy, (iv) the metrics used to evaluate the middleware, and (v) where the energy consumption is measured.

Selection process. The search and selection processes considered primary studies published until June 2022. In the automated search process, the search string has undergone minor changes to make it compatible with the specificities of each database engine. Next, the automated search procedure was performed over each electronic database according to the adapted search string. The search procedure was limited to title, abstract, and keyword fields.

Fig. 3.1 depicts the steps to select the relevant primary studies. The automated search procedure over the five electronic databases retrieved 237 results. Duplicates were removed, thus resulting in a set with 167 studies. These studies were submitted to a preliminary selection based on the selection criteria applied to the title, abstract, and keywords. Whenever the relevance of a study was in doubt, both the introduction and conclusion sections were also analyzed. This resulted in a set of 114 studies. ECs played a significant role in removing studies that would not be relevant for this SLR. Forty-five results were excluded for being unrelated to IoT or middleware (EC1), and eight results were excluded because they did not represent a primary study (EC4). No results were removed for the other ECs.

The snowballing technique [Jala12], which uses the reference list of a study or citing references to identify additional studies, was applied to each full-read selected study to retrieve additional studies. The snowballing resulted in the inclusion of one additional study. Another additional study identified by experts[6] and not retrieved in the previous steps was included in the set of potentially relevant studies. The remaining 116 studies were analyzed in full regarding their conformance to the selection criteria of this SLR. The final set contained 22 studies selected to fill out the data extraction sheet. The number of relevant studies seems to be low since energy efficiency and energy-awareness in IoT middleware are recent topics but worthwhile investigating

[6] An expert is a researcher with experience in both IoT middleware and energy-efficiency strategies.

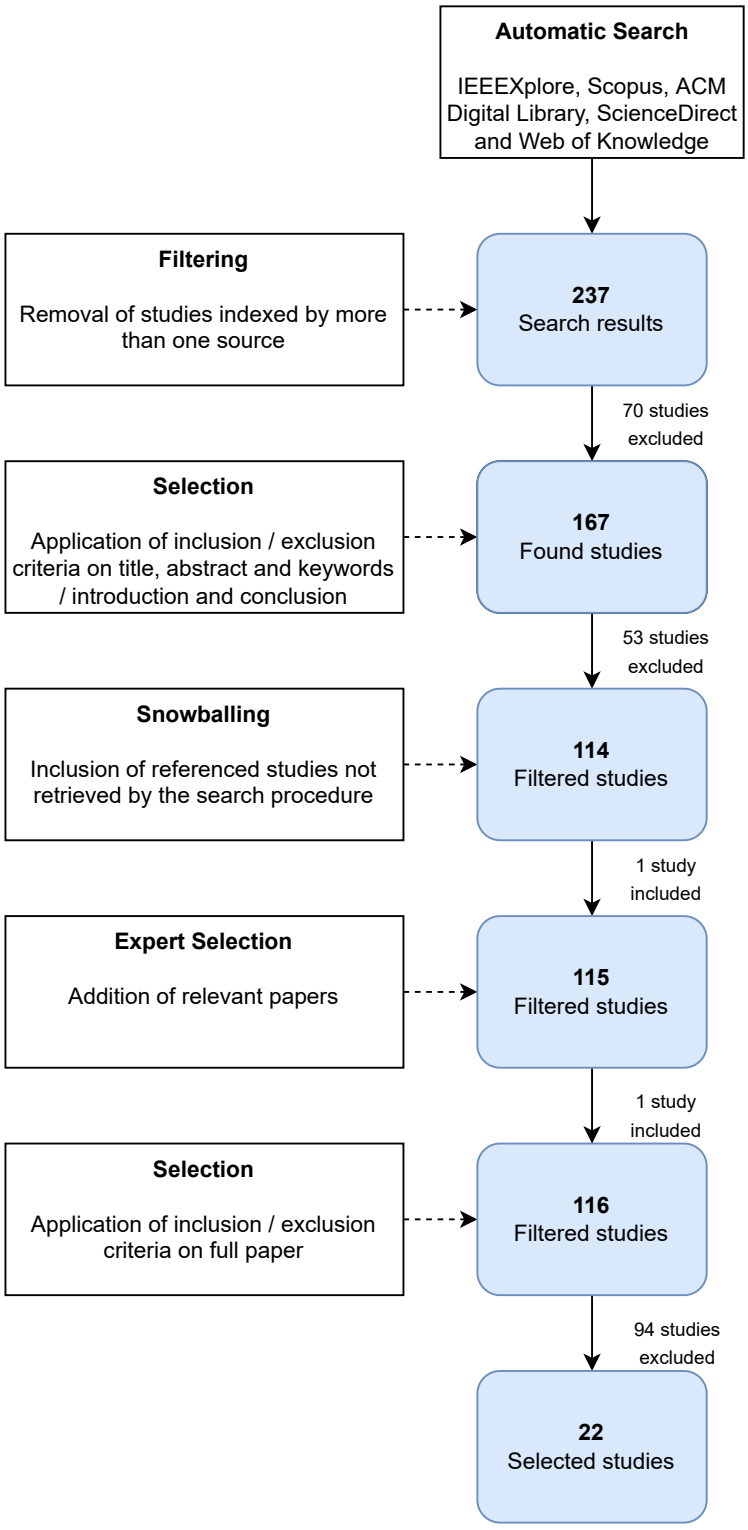


Figure 3.1. Steps to select the relevant primary studies

for future research. Table 3.1 lists the selected studies, identified as S1 to S22. Interested readers can go to Appendix III for a summary of each selected study.

Table 3.1. List of selected studies

ID	Reference	Citation count (Google Scholar, Jul. 2022)
S1	Aazam, M., Islam, S.U., Lone, S.T., Abbas, A.: Cloud of Things (CoT): Cloud-Fog-IoT task offloading for sustainable Internet of Things. IEEE Transactions on Sustainable Computing **7**(1), 87–98 (2022) [Aaza20]	8
S2	Song, Z., Le, M., Kwon, Y.-W., Tilevich, E.: Extemporaneous micro-mobile service execution without code sharing. In: Proceedings of the 37th IEEE International Conference on Distributed Computing Systems Workshops, pp. 181–186. IEEE, USA (2017) [Song17]	3
S3	Kalbarczyk, T., Julien, C.: Omni: An application framework for seamless device-to-device interaction in the wild. In: Proceedings of the 19th International Middleware Conference, pp. 161–173. ACM, USA (2018) [Kalb18]	7
S4	Al-Roubaiey, A., Sheltami, T., Mahmoud, A., Yasar, A.: EATDDS: Energy-aware middleware for wireless sensor and actuator networks. Future Generation Computer Systems **96**, 196–206 (2019) [AR19]	7
S5	Akkermans, S., Bachiller, R., Matthys, N., Joosen, W., Hughes, D., Vučinić, M.: Towards efficient publish-subscribe middleware in the IoT with IPv6 multicast. In: Proceedings of 2016 IEEE International Conference on Communications. IEEE, USA (2016) [Akke16]	23

S6	Shekhar, S., Chhokra, A., Sun, H., Gokhale, A., Dubey, A., Koutsoukos, X.: URMILA: A performance and mobility-aware fog/edge resource management middleware. In: Proceedings of the 22nd IEEE International Symposium on Real-Time Distributed Computing, pp. 118–125. IEEE, USA (2019) [Shek19]	11
S7	Jeon S., Jung, I.: Mint: Middleware for cooperative interaction of things. Sensors **17**(6) (2017) [Jeon17]	31
S8	Cecchinel, C., Fouquet, F., Mosser, S., Collet, P.: Leveraging Live Machine Learning and Deep Sleep to support a self-adaptive efficient configuration of battery powered sensors. Future Generation Computer Systems 92, 225–240 (2019) [Cecc19]	10
S9	Padhy, S., Chang, H.-Y., Hou, T.-F., Chou, J., King, C.-T., Hsu, C.-H.: A middleware solution for optimal sensor management of IoT applications on LTE devices. In: J.-H. Lee, S. Pack (eds.) Quality, Reliability, Security and Robustness in Heterogeneous Networks, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 199, pp. 283–292. Springer International Publishing, Switzerland (2017) [Padh17]	0
S10	Oliveira, E.A., Delicato, F., Mattoso, M.: An energy-aware data cleaning workflow for real-time stream processing in the Internet of Things. In: Anais do IV Workshop de Computação Urbana, pp. 71–83. SBC, Brazil (2020) [dO20]	1
S11	S. Pasricha.: Overcoming energy and reliability challenges for IoT and mobile devices with data analytics. In: Proceedings of the 31st International Conference on VLSI Design and 17th International Conference on Embedded Systems, pp. 238–243. IEE, USA (2018) [Pasr18]	4

S12	Sarkar, C., Rao, V.S., Venkatesha Prasad, R., Das, S.N., Misra, S., Vasilakos, A.: VSF: An energy-efficient sensing framework using virtual sensors. IEEE Sensors Journal **16**(12), 5046–5059 (2016) [Sark16]	29
S13	Aazam, M., Zeadally, S., Feo Flushing, E.: Task offloading in edge computing for Machine Learning-based smart healthcare. Computer Networks **191** (2021) [Aaza21]	19
S14	Mukherjee, A., Dey, N., De, D.: EdgeDrone: QoS aware MQTT middleware for mobile edge computing in opportunistic Internet of Drone Things. Computer Communications **152** (2020) [Mukh20]	35
S15	Li, W., Delicato, F.C., Pires, P.F., Lee, Y.C., Zomaya, A.Y., Miceli, C., Pirmez, L.: Efficient allocation of resources in multiple heterogeneous wireless sensor networks. Journal of Parallel and Distributed Computing **74**(1), 1775–1788 (2014) [Li14]	108
S16	Podnar Zarko, I., Antonic, A., Pripužic, K.: Publish/subscribe middleware for energy-efficient mobile crowdsensing. In: Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication, pp. 1099–1110. IEEE, ACM (2013) [PZ13]	46
S17	Al-Madani, B., Shahra, E.: An energy aware plateform for IoT indoor tracking based on RTPS. Procedia Computer Science **130**, 188–195 (2018) [AM18]	0
S18	Banouar, Y., Monteil, T., Chassot C.: Analytical model for adaptive QoS management at the middleware level in IoT. In: Proceedings of the 2017 IEEE Symposium on Computers and Communications, pp. 1201–1208. IEEE, USA (2017) [Bano17]	9
S19	Huang Z., Lin, K.J., Han L.: An energy sentient methodology for sensor mapping and selection in IoT systems. In: Proceedings of the 23rd IEEE International Symposium on Industrial Electronics, pp. 1436–1441. IEEE, USA (2014) [Huan14]	14

S20	Pasricha S.: Overcoming energy and reliability challenges for IoT and mobile devices with data analytics. In: Proceedings of the 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems, pp. 238–243. IEEE, USA (2018) [Pasr18]	4
S21	Ramachandran, G.S., Proença, J., Daniels, W., Pickavet, M., Staessens, D., Huygens, C., Joosen, W., Hughes, D.: Hitch Hiker 2.0: A binding model with flexible data aggregation for the Internet-of-Things. Journal of Internet Services and Applications **7** (2016) [Rama16]	13
S22	Elhabbash, A., Elkhatib, Y.: Energy-aware placement of device-to-device mediation services in IoT systems. In: H. Hacid, O. Kao, M. Mecella, N. Moha, H.Y. Paik (eds.) Service-Oriented Computing. Lecture Notes in Computer Science, vol. 13121, pp. 335–350. Springer Nature Switzerland AG, Switzerland (2021) [Elha21]	0

3.1.2 Results and discussion

This section summarizes and discusses the results of the SLR considering the four RQs introduced in Section 3.1.1 and the data extracted and synthesized from the analyzed primary studies. Answers to each RQ concern strategies, abstractions, evaluation, and target of the IoT middleware.

Energy-aware and energy-efficient strategies in IoT middleware

The analysis of data extracted from the selected primary studies allowed identifying several strategies and techniques for energy efficiency in IoT middleware to answer RQ1. Table 3.2 lists these strategies and techniques.

The main strategies used in the studies for providing energy efficiency and awareness are *network adaptation* (9/22) and *task offloading* (8/22). Network adaptation refers to introducing new protocols or modifying existing ones, making network optimizations (e.g., choosing the network technology), and reducing network usage at the middleware level. As examples of network adaptations, study S4 modifies the Data Distribution Service (DDS™) protocol [PC03] to improve energy efficiency, while studies S5, S9, and S12 propose

Table 3.2. Strategies and techniques for energy efficiency in IoT middleware

Strategies and techniques	Studies																					
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22
Network adaptation	×	×	√	√	√	×	√	×	√	×	×	√	×	√	√	×	×	√	×	×	×	×
Task offloading	√	√	×	×	×	√	×	×	×	×	√	×	√	×	×	×	×	×	√	√	×	√
Active node selection	×	×	×	√	×	×	×	√	×	×	×	√	×	×	×	×	×	×	×	×	×	×
Machine Learning	×	×	×	×	×	×	×	√	×	×	×	×	√	×	×	×	×	×	×	√	×	×
Data filtering	×	×	×	×	×	×	×	×	×	√	×	×	×	×	√	√	√	×	×	×	√	×

a protocol adaptation by using new algorithms. In study S5, the capabilities of IPv6 multicast are used with a publish-subscribe interaction, resulting in lower network and energy consumption. Task offloading means using the network to transfer processes or data to other locations. For example, an application running on a mobile phone could send data to a server or other mobile phones for data processing purposes. Studies S1, S2, and S6 propose offloading processes or code to other parts of the system (such as servers and fogs) to save energy on some nodes. Similarly, study S11 runs a virtual machine in a cloud environment to execute applications instead of running them on local network devices.

Active node selection is a strategy used by IoT middleware for sensor networks (3/22). In active node selection, a given node (connected objects, cloud, fog, etc.) can be selected to process data, and this node can change when needed to extend the lifetime of the entire IoT system. This refers to balancing the energy usage across multiple objects in the same network and knowing the energy capabilities of each object. Therefore, the system will select the ones with the highest energy available to process data, thus increasing the time the system will be fully alive (with all nodes working). Studies S8 and S12 propose changing the number of active nodes in a sensor network according to observations sent by these nodes and the overall communication (data) over the network. In contrast, study S4 proposes simulating the energy available in each node of a sensor network and periodically selecting the active ones.

Machine Learning is used in 3/22 studies to build a model based on input data towards making adaptations to save energy. These data can be made of information such as network conditions, CPU load in devices, amount of device communication, data sent by devices, etc. Building the model allows

for adaptations to the application's behavior to save energy or allow the model to improve and better manage data to provide the expected energy saving. Study S8 uses Machine Learning to increase the energy efficiency of the entire network and the lifespan for long periods. It can do it by determining an optimal configuration of sensors for extending their battery life using sensor sampling frequency and network usage data.

The *data filtering* capability offered by some middleware (5/22 studies) proposes processing data to reduce the amount of data transmitted or to limit the size of the message according to specific criteria (network load, CPU usage, energy usage, etc.). Studies S10, S15, S16, S17, and S21 use a data filtering strategy to continuously remove unused data to reduce the amount of data to be processed/transmitted, thereby reducing energy consumption in the system. For example, study S10 provides an energy-aware data collection that can reduce the amount of data processed and sent to the network while maintaining accurate data flow for applications that need high QoS, such as real-time applications.

Main findings (RQ1). Several strategies and techniques have been proposed in the literature for energy efficiency in IoT middleware. Most studies have focused on performing adaptations at the network level or offloading tasks. Moreover, most studies use only one strategy, i.e., combining different strategies and techniques has not been much addressed.

Abstractions for energy-awareness or energy efficiency in IoT middleware

RQ2 concerns identifying any synergy between applications (i.e., at the software level) and IoT middleware when considering energy-awareness or energy efficiency. Only four studies (S2, S3, S7, and S20) focus on providing ways for an IoT application to easily use energy-awareness (S2) or energy efficiency (S3, S7, and S20) strategies with the middleware through programming abstractions. Study S2 proposes an energy-aware abstraction, while studies S3, S7, and S20 propose energy-efficient abstractions. These proposals are presented as libraries, frameworks, or well-defined APIs abstracting away the specificities needed for an energy-aware/efficient implementation from the developer.

In study S2, application developers should provide their energy constraints for component offloading (i.e., minimum battery level) through a configuration file, and application developers and the middleware calculate the energy requirement for choosing the hosts suited for offloading. In study S3, an asynchronous API is proposed to send and receive data to multiple peers. Appli-

cations choose between two methods that implement different strategies for energy efficiency purposes: either to periodically distribute small context data to all their neighbors or to transfer a large volume of data to specific peers at one time. The middleware transparently chooses the most efficient device-to-device wireless network technology (e.g., Wi-Fi or Bluetooth) in terms of energy for the application's use. In study S7, the middleware provides a high-level API for developing IoT device applications. The middleware transparently chooses the most efficient communication strategy and adapts the application's behavior according to the data received from neighboring devices.

The solutions providing energy-aware/efficient abstractions (S2, S3, S7, and S20) are based on neighboring devices that periodically send data that are used or analyzed by the middleware to manage the energy-awareness/efficiency in the application. The studies work with those data to better communicate between the application and neighboring devices.

Main findings (RQ2). Most studies do not consider any energy-aware/efficient abstraction, leading to a concern about how easily an IoT application can enable an energy-efficient/aware strategy through middleware.

Evaluation of energy-aware/efficient IoT middleware

RQ3 is interested in identifying how the energy efficiency of IoT middleware and IoT systems built with their support have been evaluated through experimentation or simulation. All the studies concerned evaluating the IoT middleware regarding energy-awareness/efficiency. Seven of the selected studies considered using simulation to evaluate the energy consumption of the middleware. In contrast, fifteen studies present an evaluation based on testbeds and scenarios that use real-world sensors and experiments based on concrete implementations. This indicates a tendency to experiment with concretely implemented solutions gathering the information from real-case scenarios.

The data extraction process also provided information regarding the current state of the middleware, whether the middleware is implemented and working in an IoT system or an abstract architecture presents it with no concrete implementation. Nineteen of the selected studies described an implemented middleware deployed and used in a real-world scenario. In contrast, three studies explained the middleware and proposed models without a concrete implementation in IoT systems. Fig. 3.2 shows the correlation between the evaluation methods and the state of the proposed energy-aware/energy-efficient middle-

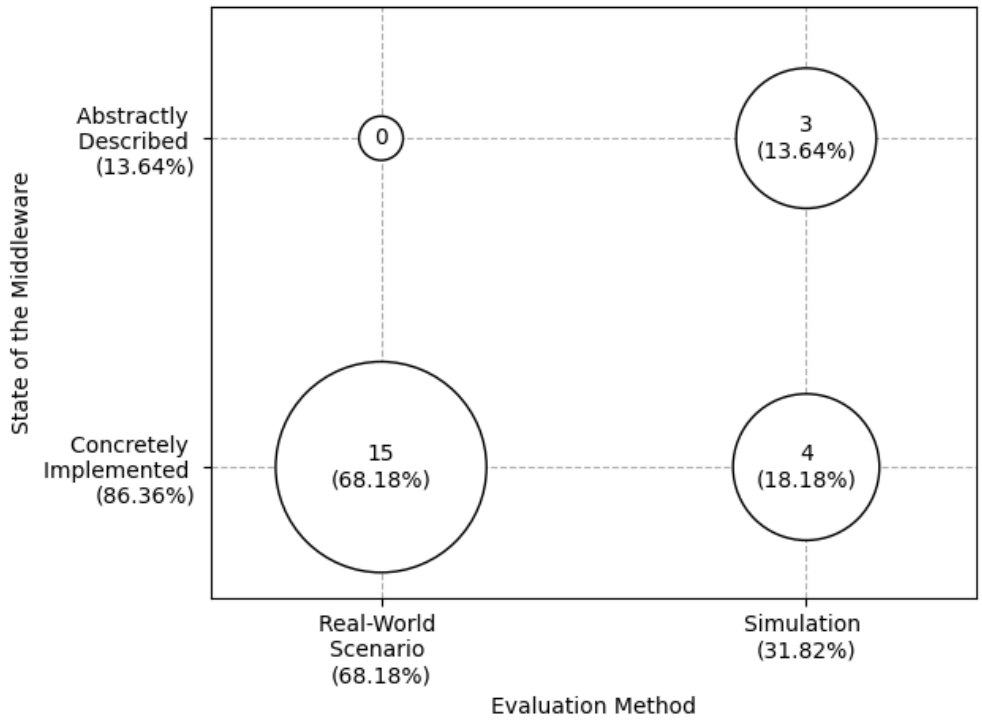


Figure 3.2. Evaluation method and state of the implementation

ware proposed in the selected primary studies. It is possible to notice that most of the concretely implemented middleware has been experimented in real-world scenarios, while the proposals abstractly described have been evaluated via simulation.

Main findings (RQ3). Most solutions present a concrete implementation and can be used in real-world scenarios with IoT systems. In contrast, there are few studies addressing evaluation with simulation.

Target of energy efficiency

RQ4 aims to identify the target of the energy efficiency of the IoT middleware. Four main targets were observed: (i) *end-user application*, representing the device used by the user to access an IoT application; (ii) *connected objects*, such as sensors and actuators; (iii) *server*, referring to middleware usually deployed in fogs or clouds; and (iv) *sensor network*, related to techniques to reduce energy consumption over the whole sensor network. Table 3.3 summarizes the

Table 3.3. Target of energy efficiency in IoT middleware

Energy efficiency target	Studies																					
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22
End-user application	×	√	√	×	√	√	×	×	×	√	√	×	√	×	×	√	×	×	×	√	√	×
Connected objects	×	×	×	×	×	×	√	√	√	√	×	×	×	√	√	×	√	×	√	×	×	√
Server	√	×	×	×	×	×	×	×	×	×	√	×	×	×	×	×	×	√	×	×	×	×
Sensor network	×	×	×	√	×	×	×	√	×	×	×	√	×	×	√	×	√	×	√	×	×	×

target of the selected primary studies concerning energy efficiency.

The middleware presented in studies S1 and S11 saves energy at the server level. For instance, the one described in study S1 has data processed at the cloud/fog level, and only the necessary data is sent to the server. Ten other studies do it on the end-user application side, and nine studies focus on the connected object side. For example, study S8 proposes a deep sleep of connected objects to save network usage. Studies S4, S8, S12, S15, S17, and S19 propose to reduce the energy consumption in the sensor network, e.g., by changing communication among nodes in the same network.

In the selected studies, the middleware in the IoT system is deployed in different locations (see Fig. 3.3). Studies S2, S9, S11, S13, and S20 propose deploying the middleware at the end-user application side. On the other hand, studies S3, S6, and S16 propose a middleware deployed in both the end-user applications and the cloud, while studies S7, S12, S15, S19, and S21 deploy the middleware in the connected objects. It is also possible to notice that servers have been used to deploy the middleware without being the target of energy efficiency in the proposals.

Studies S4, S8, S17, and S22 present middleware on the cloud side. In contrast, studies S1, S5, S10, and S18 present middleware deployment in a gateway, whereas study S14 deploys its middleware in the gateway and the connected object. The choice to deploy the middleware on the cloud, gateway, or other parts of a system is highly influenced by the implemented solution, and there is no rule to choosing the right place. Furthermore, studies S3, S6, and S16 deploy their middleware in multiple places so that the placement of the middleware is related to how the middleware energy-aware/efficient strategy works.

Main findings (RQ4). Most of the selected studies present solutions working on the end-user application side, thereby showing a more significant

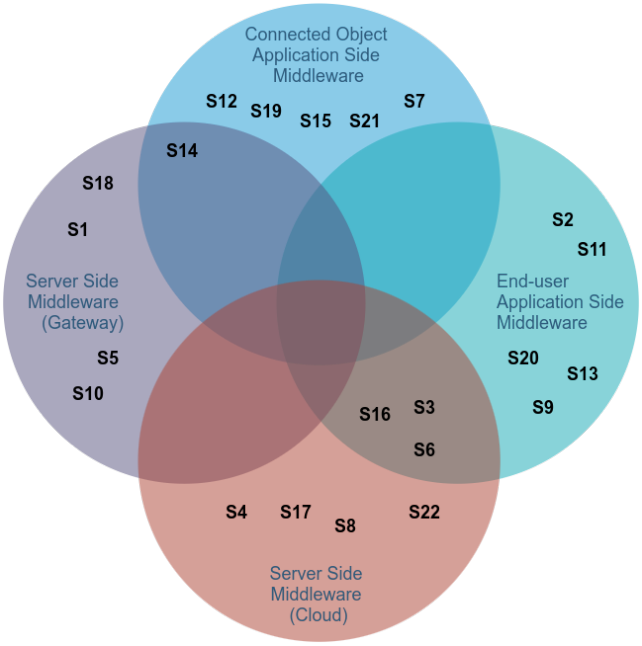


Figure 3.3. Middleware deployment locations addressed in the selected primary studies

concern related to the lifetime of battery-constrained devices (e.g., smartphones). A few studies concern more than one target, leading to concerns regarding the distribution of energy awareness and efficiency in an IoT system.

3.1.3 Summary

The IoT paradigm offers promising opportunities to improve daily life through continuous, dynamic cooperation between systems and physical objects. However, properly dealing with the development of IoT systems means providing solutions to significant challenges such as energy efficiency and resource consumption while facilitating development. Addressing the energy-efficiency challenge at the middleware level can reduce the energy consumption of systems supported by that middleware while relieving application developers from this concern. Furthermore, energy-awareness in middleware will impact the efficiency of the applied strategies as there will be more context about where and how the energy is being used, providing valuable knowledge for lowering the energy usage through middleware.

This section has presented the results of an SLR to identify and understand

essential features for energy efficiency and energy-awareness in IoT middleware. The systematic selection and analysis of 22 studies systematically selected and analyzed have shown increasing concern for energy efficiency in IoT middleware. However, a few works consider energy-related issues from the perspective of developing IoT applications. Considering that IoT middleware is the layer that links IoT applications to other components of an IoT system, the abstraction of energy-efficiency/awareness inside IoT applications is an issue that needs to be further developed.

3.2 Energy-efficiency in HTTP and MQTT protocols

The usage of IoT protocols through many IoT software in the many layers of an IoT system have an impact over the energy consumption that can be vital to IoT applications. Having a better view of how the handling of the data exchanged between IoT protocols can widen the knowledge of application developers to have care more about how much energy their code is consuming.

In order to propose an energy efficient middleware between IoT consumer applications and IoT platforms, we have to study the energy efficiency of the protocols used for those interactions. As shown in Section 2.4.3, and in Disdarevi et. al [Dizd19], the protocols commonly used in this context are MQTT [OASI15] and HTTP [Niel99].

Regarding HTTP and MQTT, an analysis of the related works concerning their energy efficiency was done. Four research papers were selected that include energy consumption measures in the evaluation of HTTP and/or MQTT. We have to mention that, to the best of our knowledge, the number of papers on this subject is low and the measures do not isolate the consumption on the consumer side. Furthermore, none of them study the impact of the interaction pattern on the energy consumption.

Section 3.2.1 presents briefly the HTTP and MQTT protocols and Section 3.2.2 presents the analysis or the related works.

3.2.1 HTTP and MQTT overview

HTTP (HyperText Transfer Protocol) is the footing of the client-server architecture in the Web [Niko20]. It is a standard working over TCP/IP that can be used to provide data of any kind of data, for example, XML or JavaScript

Object Notation (JSON) in payloads from IoT objects to applications and vice-versa. When it comes to using the protocol, it is mostly used in its Request/Reply interaction pattern. However, as an example, the FIWARE platform also uses HTTP for the publish-subscribe interaction pattern, where the client is an open listener and the server posts available data.

MQTT is a lightweight protocol with the publish/subscribe interaction pattern. It originated in 1999 proposed by IBM [7] and was standardized by OASIS [8] in 2013. It is an open standard protocol that can work over TCP/IP and involves three main components: publisher, subscriber and broker. Publishers and subscribers exchange messages. Brokers are responsible to filter incoming messages and distribute them properly according to the message topics. MQTT implements three different models of message exchange known as Quality of Service, where the delivery with QoS 0 being at most once, QoS 1 being at least once, and QoS 2 being exactly once.

HTTP and MQTT are both high-level protocols that work at the application level and enable applications to communicate over a network. They have different ways of providing this communication and their usage by IoT applications is dependent of what are the requirements. On one side HTTP has been used for a long time and is well structured, coming with multiple features that could facilitate the development of communication over the internet. On the other hand, MQTT has shown to be a good protocol with very little overhead as it allows for messages to structure themselves leaving this task for IoT applications to marshal/unmarshal data received.

3.2.2 Synthesis of related works

A synthesis of the study is presented in Table 3.4. For those related works, the following points have been analyzed. Since our objective is to study the consumer side of an IoT architecture, we indicate whether the study is conducted on the producer side (P), on the consumer side (C), or on both sides. We mention which IoT protocols were compared in the work. We also indicate the experimental conditions: the device where the measure was conducted and the type of network. The last aspect concerns the type of the evaluation, analytical or experimental and, if experimental, the tool they have used to measure the energy consumption.

Bandyopadhyay and Bhattacharyya present an analysis of MQTT and CoAP [Band13]. They examine the resource usage including energy consump-

[7] https://www.ibm.com/
[8] https://www.oasis-open.org/

tion according to the message size and the packet loss ratio. The energy consumption of the most reliable configurations was measured on a Wide Area Network for the protocols CoAP and MQTT with QoS 2. They show that with a perfect network without any loss, MQTT with QoS2 is more than ten times more consuming than CoAP. Concerning energy efficiency, they only study MQTT with QoS2. They do not define whether the measures are done on the producer or/and consumer side, which makes it difficult to know which side of the architecture was studied. They also do not mention how the energy consumption was measured.

Toldinas et al. perform a dedicated study of MQTT QoS levels and their energy consumption [Told19]. They use a ESP-WROOM-02 hardware device connected to the network through Wifi 802.11 and acting both as producer and consumer. For each level of QoS, the remaining battery voltage level was measured using a digital multimeter as an indicator of energy consumption. This study provides a good indication of the percentage increase in energy consumption for each level of the QoS compared to the previous one. However, it does not allow effective energy-consumption conclusions to be drawn about the behavior of a consumer or a producer as the same device is used for both tasks.

Hofer and Pawaska studied the impact of MQTT and HTTP protocols on CPU, RAM, and energy consumption [Hofe18]. The device used is a Raspberry Pi connected by Ethernet, that acts both as a producer and a consumer, as a consequence they can not isolate the energy consumption on the consumer side. They do not mention what QoS was used for MQTT. For the energy evaluation, the authors studied the Ampere per second in the device using an oscilloscope. The study proved that MQTT outperformed HTTP RESTful in terms of data overhead which is the amount of extra data needed to be sent to a client (e.g HTTP Headers, MQTT headers, etc), a nearly four times higher throughput. Furthermore, MQTT also had lower resource consumption and significantly lower energy consumption. As HTTP is used with the synchronous interaction pattern and MQTT is used with the publish/subscribe interaction pattern, it is not possible to isolate the impact of the interaction pattern from the impact of the protocol.

Joshi et al. presented a comparison in terms of protocol impact on throughput and battery consumption between MQTT (QoS not specified), CoAP and HTTP RESTful [Josh17]. The device used was a Raspberry Pi, which acted only as a producer. For energy consumption, the percentage of battery consumption per hour was taken as a reference. However, it was not mentioned how it was calculated. The conclusions are: (i) HTTP consumes more energy

than MQTT, and (ii) with the same amount of battery it is possible to send 100 times more messages with MQTT compared to HTTP. Although the work was not dedicated to the study of energy consumption, it lacks details on how the measures were implemented as well as the conditions of the experiment (e.g. network type).

Ref	Network	P/C	MQTT QoS			HTTP		Evaluation
			0	1	2	Sync	Pub/sub	
[Band13]	WAN em.	?	×	×	√	×	×	?
[Told19]	Wifi	P+C	√	√	√	×	×	simulation
[Hofe18]	Ethernet	P+C	?	?	?	√	×	oscilloscope
[Josh17]	Wifi	P	√	×	×	√	×	calculated

Table 3.4. Synthesis of the related work

This related work study shows that there is a lack of experiments for isolating the energy consumption of HTTP and MQTT on consumer applications. We will present our contribution on this subject in Chapter 5.

3.3 IoT middleware for consumer applications

The IoT research area is wide with protocols, standards, and platform specifications. Software developers need to learn many technologies while being careful with application resource usage. In that context, several works propose generic abstractions in order to ease the development of IoT applications. We organize the related work presentation in three parts: Domain Specific Language (DSL) (Section 3.3.1), Application Mashup (Section 3.3.2), IoT middleware (Section 3.3.3).

3.3.1 Domain Specific Language

In order to provide application abstractions for non developer specialists, many domain specific dedicated programming languages are used nowadays. More specifically, some middleware solutions provide their own *Domain Specific Language (DSL)* to provide abstractions for building IoT applications. Boujbel and co-authors [Bouj14] present MuScADeL (MultiScale Autonomic Deployment Language), a domain specific language dedicated to multiscale and autonomic IoT system deployment. MuScADeL allows designers to declare multiscale deployment properties without exact knowledge of the deployment domain, pro-

viding for example the deployment of mass market smart applications over the IoT. Similarly, Delicato and co-authors [Deli13] propose the Programming and Execution Module (PEM) solution that provides a Web Mashup DSL specifically tailored for Wireless Sensor Networks (WSNs), as well as an interpreter for this DSL. The Web Mashup allows *ad-hoc* Web applications to be built upon the combination of real-time information (data, presentation, and functionality) through the composition of available services such as publishing and discovering the capacities of available WSNs.

DSLs allow developers to generate useful and repetitive lines of codes, however developers have to learn new languages and integrate new development tools, and keep limited to the DSL goals.

3.3.2 Application Mashup

Using *open application programming interfaces (API)*, Web Mashup favors easy and fast integration of data sources to produce augmented results that were not considered when producing raw data [Bens08]. One example of an IoT Web Mashup is Wirecloud [WIRE21], an end-user Web-centered application mashup platform aimed at allowing end users, with no programming skills, to easily create Web applications and dashboards/cockpits, e.g. to visualize their data of interest or to control their smart home or environment. Node-RED [NODE21] is another Web application mashup that allows wiring together hardware devices, APIs, and online services. It provides a Web browser-based flow editor, which can be used to create JavaScript functions. Elements of applications can be saved or shared for re-use.

Application mashups have advantages of a user-friendly graphical development that comes with fast composition, however they are limited to the available graphical components that may rapidly limit the IoT application designer.

3.3.3 IoT middleware

IoT applications require the use of a great number of technologies (IoT protocols, IoT platforms, IoT devices). A manner to abstract away the complexity is to use a middleware and the Proxy pattern [Diaz08]. Soundararajan and Robert [Soun08] show how the Proxy pattern can help to implement a benchmarking application. They provide a client-side Proxy pattern that hides the complexity of protocols and encapsulates the knowledge of how to contact the servers. Similarly, Sutra and co-authors [Sutr17] have built the CRESON sys-

tem that creates a Proxy at the client side in order to contact databases over the Internet. It abstracts the communication protocols involved while preparing to transfer and process data required by the application.

3.3.4 Synthesis

Several methods can be considered for masking the complexities of building IoT consuming applications. DSLs and Mashup are promising ones, however they may limit the development and deployment of the applications. Proposing a middleware is the approach that we adopt in this PhD. We present in Chapter 4 the IoTvar middleware that propose taming IoT consuming application development through the paradigm of IoT variables handled by specific proxies.

3.4 Conclusions

Building energy efficient IoT consuming applications is a difficult task. In this related work chapter we have highlighted the need for high level programming abstractions for IoT application developers. As shown by the SLR (Section 3.1), a low amount of IoT middleware consider programming abstractions, which could lead to a high complexity to use IoT specific middleware. This complexity could also extend to the IoT protocols being used (e.g HTTP and MQTT) and worsen the time necessary to develop software. Furthermore, there are energy efficiency/awareness concerns to be taken into account by the developer. In this PhD, our contributions are the proposition of high level abstractions for developing IoT consuming applications, made available through an IoT middleware. The middleware layer is the level where we propose energy efficient and energy aware mechanisms. We take into account the *Network adaptation* mechanism to provide strategies for energy-efficiency/awareness in IoT middleware for consumer applications and show these strategies in Chapter 6.

Part II

IoT Middleware and Energy-Efficiency: Contributions

Chapter 4

IoTVar

This chapter presents the first contribution of the Ph.D. that aims at answering two of the research questions introduced in Section 1.2: *(RQ1)* How can a middleware support common abstractions for interacting with IoT platforms? *(RQ2)* What is the cost, in terms of CPU usage, memory usage, and energy consumption, of abstracting IoT platforms using an IoT middleware? The objectives are (1) to lower the learning and development costs of developing an IoT consumer application, while (2) maintaining the cost of the middleware itself reasonable in terms of additional CPU, memory, and energy.

For those purposes, we propose IoTvar [Borg19], an IoT middleware designed to abstract the interactions of consumer IoT applications with context management IoT platforms through the proxy design pattern [Shap86]. IoTvar allows developers to declare IoT variables that enable the discovery of data-producer objects virtualized on IoT platforms, while transparently dealing with automatic updates of those variables through transparent interactions with the IoT platforms.

In Section 4.1, we present the generic software architecture of IoTvar, designed to easily integrate new IoT platforms. Then, we present the role of the IoTvar proxies in Section 4.1.3. Section 4.2 presents the abstractions manipulated by an IoT application developer using IoTvar and evaluates the gain in terms of the line of codes to be written. Finally, Section 4.3 presents an evaluation of the IoTvar middleware in terms of additional costs (CPU, memory, energy) for three IoT platforms.

4.1 IoTVar Software Architecture

The IoTvar middleware is architecturally designed to ease the integration of new IoT platforms by exposing common interfaces. It has been implemented for the three IoT platforms analysed in Section 2.4: Orion, OM2M and muDEBS.

4.1.1 General architecture

Figure 4.1 shows the main components of the IoTvar middleware which proposes several components organized in layers.

The two bottom layers are shared by all the platforms. The *protocol layer* defines components that interact with protocols commonly used between IoT consumer applications and IoT platforms such as HTTP and MQTT; Other protocols could be added, if necessary. Then, the *interaction layer* provides two components that manage the interaction patterns, i.e. request/reply and publish/subscribe regardless of the IoT platform.

The top layers contain components specific to IoT platforms: the *API layer* and the *unmarshaller layer* act as a glue that maps the API calls and data structures to the IoT platforms; the *discovery layer* provides functionalities for discovering devices and managing filtering mechanisms.

When the middleware is started, IoTvar creates a pool of threads with size equals to the number of available processors. For the request/reply interaction pattern, each variable is scheduled to be managed by the pool of threads with a period given by the refresh time. On the other hand, the publish/subscribe pattern depends on the IoT platform and protocols used.

4.1.2 The cost of extending IoTvar

The IoTvar architecture allows the addition of further IoT platform extensions. Fig. 4.2 depicts the steps to integrate a new platform with IoTvar. If there is

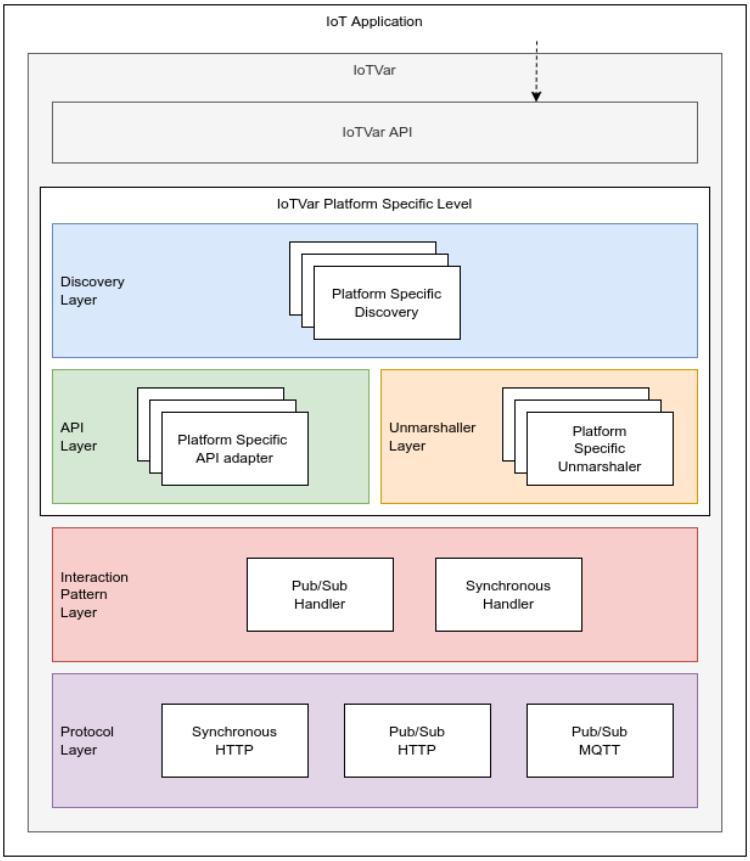


Figure 4.1. IoTvar generic architecture

no support for a protocol in use by the platform, the developer has to provide a generic implementation of this protocol. Because each IoT platform comes with its own data model and API, a specific data unmarshaller and an API adapter have to be provided.

The amount of lines of code needed to add an extension varies from platform to platform. We show, in Table 4.1, the number of lines of code necessary to implement the IoTvar components generics and specifics for the three platforms. To understand the numbers, it is important to highlight that muDEBS provides a separate library with its context data model and the unmarshaler is done outside IoTvar. Furthermore, Orion was the element that necessitated the most lines of code especially because of its discovery component, which supports many features of the API such as searching, filtering, and geolocation.

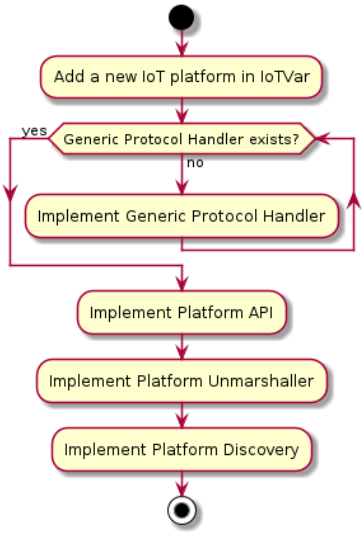


Figure 4.2. IoTvar extension steps.

Table 4.1. Number of lines of code by component

Component	Lines of code			
	generic	muDEBS	OM2M	Orion
Platform Specific Discovery	-	97	98	406
Platform Specific API adapter	-	102	198	185
Platform Specific Unmarshaler	-	-	50	153
Synchronous Handler	235	-	-	-
Pub/Sub Handler	154	-	-	-
Synchronous HTTP	139	-	-	-
Pub/Sub HTTP	238	-	-	-
Pub/Sub MQTT	202	-	-	-
Total	917	199	346	744

4.1.3 IoTVar proxies

IoT variables are linked to a proxy representing an entity of an IoT platform. The proxy activates the handler in charge of interacting with the IoT platform. The main tasks of the proxy are: (1) to discover the virtualized IoT device in the platform (2) to manage the updates with new observations, which includes unmarshalling the received data. Figures 4.4 and 4.3 show the sequence diagram of IoTvar from the creation of a variable until the data from the IoT

platform is returned for the Request/Reply and Publish/Subscribe interaction pattern, respectively.

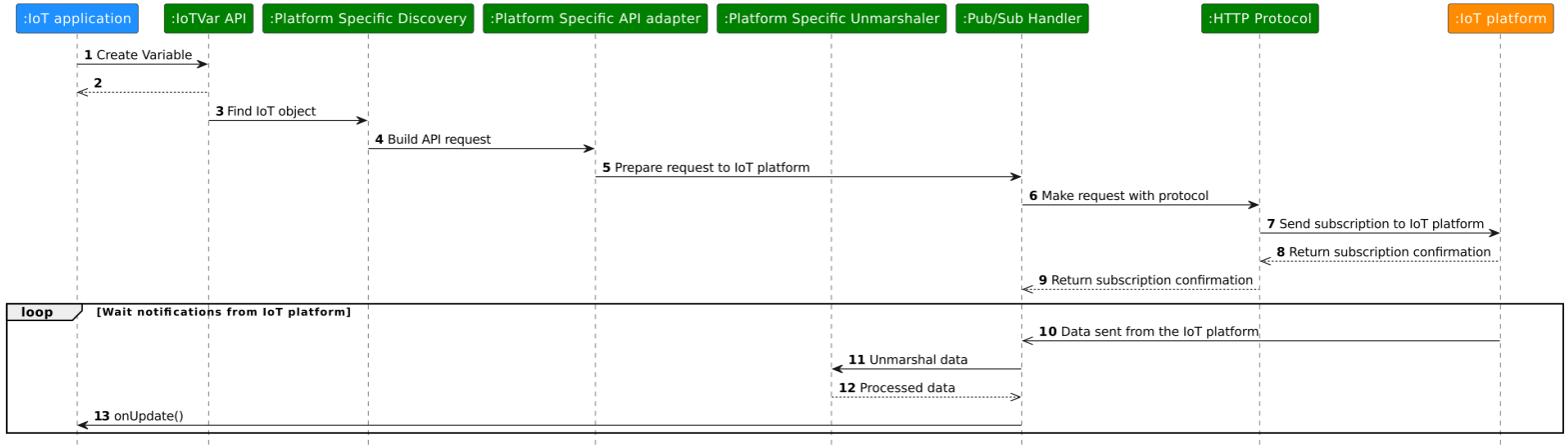


Figure 4.3. IoTvar sequence diagram for the Publish/Subscribe interaction pattern.

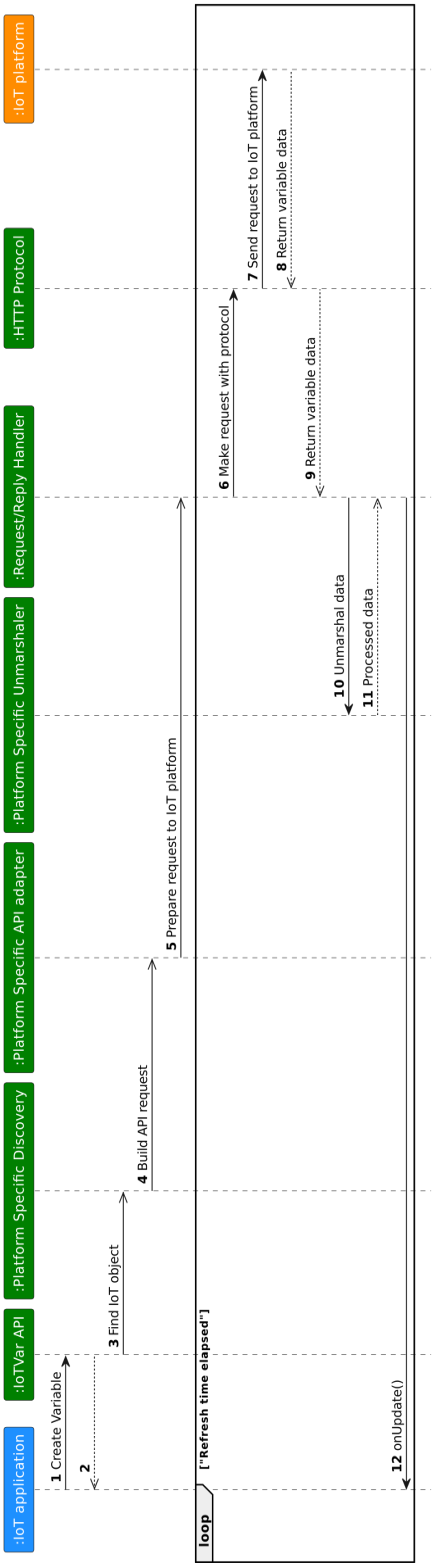


Figure 4.4. IoTvar sequence diagram for the Request/Reply interaction pattern.

4.2 IoTvar abstractions for the developer

The IoTvar middleware is available through a Java library. We show below the code to be written by a java developer to interact with an IoT platform. As an example, we rely on the case of an IoT consumer application that displays the current temperature in the vicinity of the Eiffel Tower in Paris.

4.2.1 IoTvar variable declaration

The usage of IoTvar is shown in Listings 4.1 and 4.3 with code in the JAVA programming language to display the up-to-date temperature in the vicinity of the Eiffel Tower. Listing 4.1 shows an example for the declaration of the IoT variable (Lines 1–9) and the registration of a listener for this variable (Line 10). To declare an IoT variable, the developer provides IoTvar with the identifier (Line 2) and type (Line 3) of the searched-for sensor (`Temperature` at Line 4), the location of the Eiffel Tower (Line 5), the required refresh time as a quality parameter (Line 6), the size of the local history of values (Line 7), additional filters to be provided to the platform (e.g `temperature>10;humidity>10` at Line 7), the configuration parameter of the IoT platform (Line 7), the interaction strategy with the IoT platform (Line 8), and the class to be used for unmarshalling purposes (Line 9). The location is defined by the latitude, the longitude plus the radius in meters.

Listing 4.1. Declaring a variable using IoTvar

```
1  IoTVariableFiware<Integer> temperatureEiffelTower =
2    new IoTVariableFiware<>("temperature_eiffel_tower_310", // ID
3    "LM35", // Type
4    "Temperature", // Attribute
5    new Location("location", 48.6223426, 2.4404356, 100.0),
6    new RefreshTime(10, TimeUnit.SECONDS),
7    10, null, orionConfiguration, // History size, filters and plat. config.
8    HandlerStrategy.SYNC // Or PubSub
9    Integer.class);
10 temperatureEiffelTower.registerIoTListenerP(display);
```

4.2.2 IoTvar update listener

IoTvar automatically activates a listener when the temperature observation is updated, i.e. either when a refresh has been requested (in the synchronous strategy) or when a notification has been received (in the publish-subscribe strategy). Listing 4.2 displays the basic interface for an observer with the `onUpdate` and `updateIssue` methods. The former method is called each time a new observation is provided by IoTvar, whereas the latter is called when

the update cannot comply with the specified constraints, leaving the developer with the error to be treated.

Listing 4.2. Interface of a listener

```
1 public interface IoTVarObserver {
2   void onUpdate(Observation newObservation);
3   void updateIssue(String issue);
4 }
```

Listing 4.3 presents the code of a simple observer class, `TextDisplay`, which implements the `IoTVarObserver` interface and displays the temperature around the Eiffel Tower. In this example, the observer simply logs the observation with the received temperature (Line 4), and, then, logs any error (networking error, data error, etc.) shown in Line 7.

Listing 4.3. Declaration of a listener

```
1 public class TemperatureDisplay<Meteo> implements IoTVarObserver {
2  private static final Logger logger = LogManager.getLogger(TextDisplay.
       class);
3  public void onUpdate(Observation newObservation) {
4   logger.info("Current temperature around the Eiffel Tower: " +
        newObservation);
5  }
6  public void updateIssue(String issue) {
7   logger.info("There was an error updating the sensor: " + issue);
8  }
9 }
```

4.2.3 Gains in terms of lines of codes

Table 4.2 shows the high differences for one variable regarding the number of lines of code that application developers need to write to directly use the APIs of FIWARE, OM2M, or muDEBS, or if they use IoTvar. It is clear the advantage of using IoTvar.

Table 4.2. Number of lines of code when developing with and without IoTvar

Interaction pattern	Lines of code	
	With IoTvar	Without IoTvar
Synchronous FIWARE	15	450
Publish-subscribe FIWARE	15	600
Synchronous OM2M	15	400
Publish-subscribe OM2M	15	200
Publish-subscribe muDEBS	15	450

4.3 IoTvar Evaluation

This section reports a quantitative evaluation of IoTvar and its integration within the FIWARE, OM2M, and muDEBS platforms. This evaluation involves a performance assessment that considers the same application written with and without IoTvar (i.e., directly accessing the IoT platform) aiming at measuring the overhead of the middleware in terms of CPU, memory, and energy usage. The application goal is to receive and display in the console the data that is returned from a temperature sensor around the Eiffel Tower. The tests done with this application vary the number of sensors from 25 to 200 in a 25 sensors step (25, 50, ..., 200), with a refresh time of 1 second for each sensor, and having a small local history of the 10 latest values for each sensor.

We collect the performance measurements (CPU, memory, and energy consumption) by wrapping both the method called by the synchronous handler and the method to handle notifications sent by FIWARE, OM2M, and muDEBS. This is implemented using AspectJ [Kicz01], which provides an encapsulation around the main methods of the IoTvar structure, which is woven into the IoTvar code. This is done to be able to measure CPU and memory usage of only the methods which perform the data processing and also communication without modifying the code of the middleware.

Currently, no software for energy measurement includes the consumption of the network interface in the energy consumption measurements. Some tool such as RAPL can perform energy measurements but are still limited to CPU and RAM energy consumption. Making it difficult the usage of those tools for our study. As a consequence, we decided to use a Yocto wattmeter [Yoct] to measure the energy consumption of the whole computer. The wattmeter counter is reset at the beginning of the tests, and at the end the total energy consumed is retrieved and saved in a file along with CPU and memory consumption. As the wattmeter measures the consumption of the whole computer, we have another test to get the energy consumption of the computer running without the application (idle consumption) and isolate the consumption of the application. The difference between the whole computer and idle time is then used for the analysis.

For quantitative analysis purposes, we performed hypothesis testing. To decide about the test to use, we first perform the Shapiro-Wilk test [Shap65], a powerful statistical test to verify if a sample follows a normal distribution. For the significance level $\alpha = 0.05$, we notice that the values do not follow a normal distribution, thus leading us to perform a non-parametric hypothesis test.

Thereafter, we choose Mann Whitney's U-test [Mann47], one of the most used non-parametric tests, to verify if there are differences between the two independent samples. The null hypothesis states that the means of the values in the samples are the same, i.e. there is no statistically significant difference between the samples. The t-test returns a p-value that is compared to the adopted significance level $\alpha = 0.05$. If the p-value $< \alpha$, then the null hypothesis is rejected and the alternative hypothesis is accepted, otherwise it is not possible to conclude if there is a statistically significant difference between the analyzed samples. In other words, when the returned $p-$value is less than 0.05, we reject the null hypothesis and conclude that there is a statistically significant difference in terms of using or not using IoTvar.

To better check, the measuring strength of the claims gathered from the U-test, an effect-size statistical analysis is run over the test data using the $A-$index by Delaney and Vargha [Varg00]. The analysis gives us the levels of the effect size that are interpreted with Hess and Kromrey magnitude [Hess04] that can present the following levels: (i) Negligible, (ii) Small, (iii) Medium, and (iv) Large.

In Section 4.3.1, we describe the setup for the tests with the structure and an explanation of the tests. In Sections 4.3.2 and 4.3.3, we present the results and the statistical testing for the synchronous interaction, and for the publish/subscribe interaction, respectively. Finally, in Section 4.3.4, we give an overview of the results and conclusions of the statistical analysis.

4.3.1 Setup for the experimentation

Data for each measure (CPU, memory, and energy) are collected during 30 executions of five-minute testing. The first minute of the test is the warm-up phase and it is not recorded: it ensures that the class loading is complete in the Java Virtual Machine (JVM) in order to avoid interference in the results [Geor07]. The last four minutes constitute the effective run phase.

Fig. 4.5 illustrates the setup of the tests. The computer that executes the client application consuming IoT data is a Dell notebook with an i7-8665U CPU @ 1.90GHz, 32GB ram with Debian 9 OS. The server computer is for executing the IoT platform and the IoT data producers, which simulate IoT sensors that send IoT data through the IoT platform. It has an i7-4770K CPU @ 3,9 GHz, 16 GB ram with Ubuntu 16.04 OS. Furthermore, the client computer is plugged into a YoctoPuce YoctoWatt wattmeter to collect energy-related measures. The client application communicates with the server through a local isolated WiFI network.

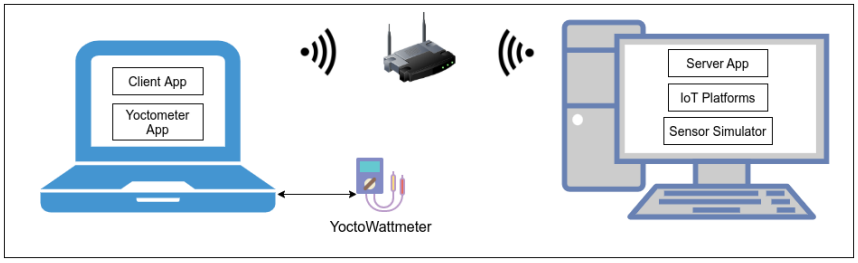


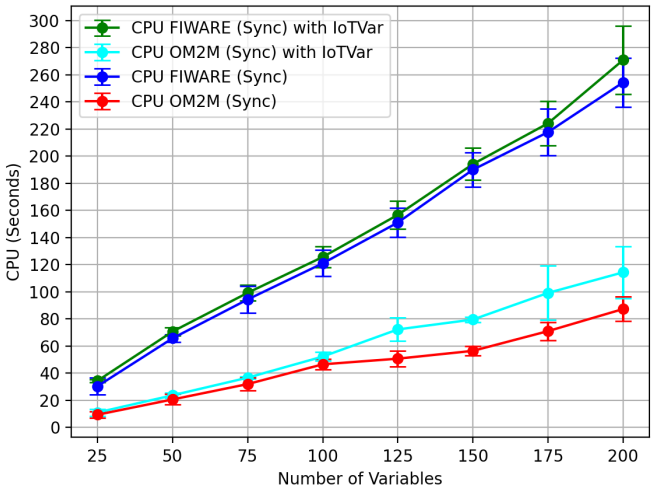
Figure 4.5. IoTvar experimental setup

The different versions of the IoT application (with or without IoTvar) get sensor data either through synchronous calls or publish-subscribe notifications. In the former interaction pattern, the IoT application with IoTvar creates an IoT variable for each sensor and the middleware sends one request per second to get data for each declared sensor. In the latter interaction pattern, the IoT application with IoTvar creates an IoT variable for each sensor, the middleware registers to the corresponding entity, and the IoT platform notifies IoTvar about all the updates.

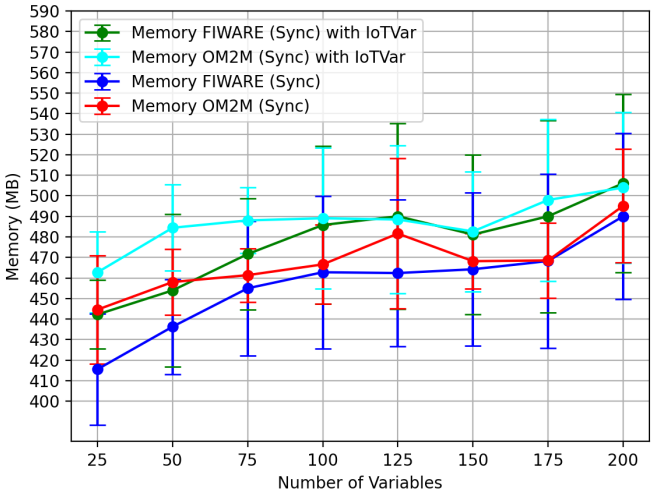
4.3.2 Results for the synchronous interaction pattern

The CPU performance of synchronous calls is displayed in Figures 4.6a for the three IoT platforms. IoTvar increases the demand for CPU and memory because more processing is necessary to handle the pool of threads and to maintain the collection of IoT variables (search, update, and maintain the history). As a consequence, the energy consumption with IoTvar is greater than without IoTvar due to the amount of overall CPU and memory usage by IoTvar in order to provide a usable abstraction to application developers. The difference between using or not IoTvar are displayed, in percentages, in Table 4.3.

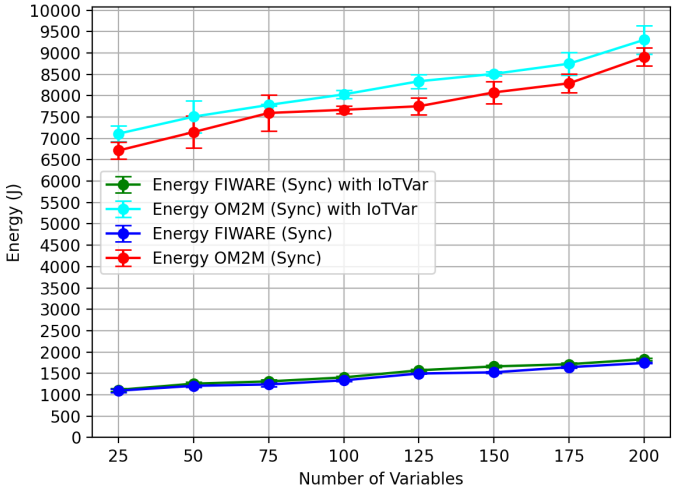
Concerning the FIWARE platform, Table 4.4 shows the results of the U-test with respect to CPU, memory, and energy. Based on the returned p-values, we reject the null hypothesis and conclude that there is a statistically significant difference for memory and energy, that is, the overhead put by the IoTvar abstraction does affect the overall performance. For CPU, we have p-values greater than 0.05 for 75, 150, and 175 variables, thus indicating that there is no statistical difference for the overhead caused by IoTvar in these cases. Furthermore, the overhead size caused by the usage of IoTvar is shown in Table 4.5 with the results from the $A-$index test. In total, there are 5 cases



(a) Synchronous CPU consumption



(b) Synchronous memory consumption



(c) Synchronous energy consumption

Figure 4.6. IoT platforms synchronous consumption with and without IoTvar

Table 4.3. Difference between using and not IoTvar for the synchronous interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU FIWARE	12.38%	7.29%	5.07%	3.79%	3.68%	2.15%	2.94%	6.29%
Memory FIWARE	6.22%	3.97%	3.6%	4.85%	5.81%	3.57%	4.55%	3.23%
Energy FIWARE	1.79%	4.08%	5.53%	4.98%	4.92%	8.67%	4.13%	4.9%
CPU OM2M	16.21%	13.92%	13.8%	11.59%	35.14%	34.02%	33.16%	26.86%
Memory OM2M	4.01%	5.61%	5.62%	4.71%	1.41%	3.05%	6.09%	1.78%
Energy OM2M	5.7%	4.91%	2.45%	4.59%	7.27%	5.23%	5.39%	4.39%

with a small impact, 10 cases with a medium impact, and 9 cases with a large impact.

Table 4.4. p-values from the U-test for the FIWARE synchronous interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	0.00231	1.68e-07	0.0892	0.0287	0.00677	0.052	0.0869	0.0189
Memory	5.09e-06	0.0034	0.00175	0.00881	0.00175	0.0113	0.00241	0.131
Energy	0.0121	0.000356	3.01e-08	3.55e-11	1.92e-11	3.14e-11	2.06e-09	3.88e-09

Table 4.5. Magnitude from the U-test for the FIWARE synchronous interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	medium	large	small	small	medium	small	small	medium
MEM	large	medium	medium	medium	medium	medium	medium	small
Energy	medium	large	large	large	large	large	large	large

Concerning the OM2M platform, Table 4.6 shows the results of the U-test with respect to CPU, memory, and energy. Based on the returned p-values, we reject the null hypothesis and conclude that there is a statistically significant difference for CPU and energy, that is, the overhead put by the IoTvar abstraction does affect the overall performance. For memory, we have p-value greater than 0.05 only for 150 variables, thus indicating that there is no statistical difference for the overhead caused by IoTvar in this case, but for the

rest of the values, the test indicated that there is an overhead. Furthermore, the overhead caused by the usage of IoTvar is shown in Table 4.7 with the results from the $A-$index test. In total, there are 2 cases with negligible impact, 1 case with small impact, 3 cases with medium impact, and 18 cases with a large impact.

Table 4.6. p-values from the U-test for the OM2M synchronous interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	1.52e-06	0.0159	0.00494	2.65e-07	0.498	0.497	3.33e-12	3.34e-11
Memory	0.00176	1.69e-05	3.57e-05	0.00681	0.197	0.0545	1.31e-05	3.34e-11
Energy	2.27e-09	1.41e-08	1.1e-05	1.18e-10	3.27e-12	5.87e-10	3.31e-12	8.56e-05

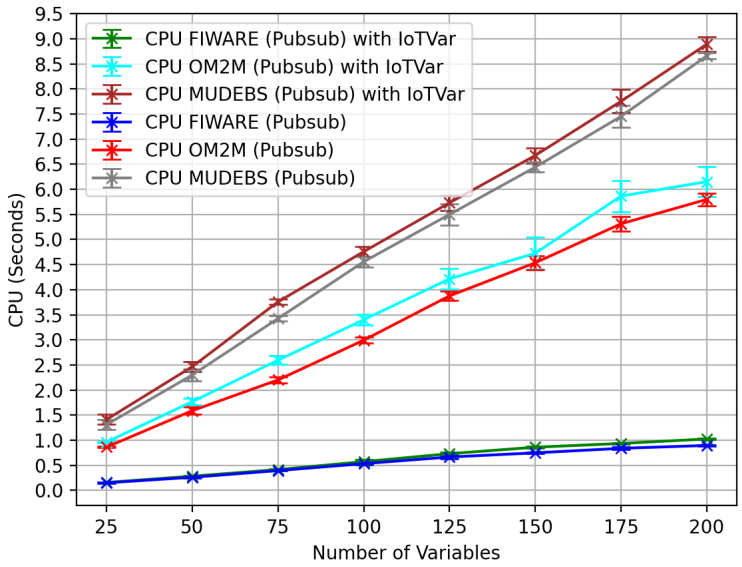
Table 4.7. Magnitude from the U-test for the OM2M synchronous interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	large	medium	large	large	large	large	large	large
MEM	medium	large	large	medium	negligible	small	large	negligible
Energy	large	large	large	large	large	large	large	large

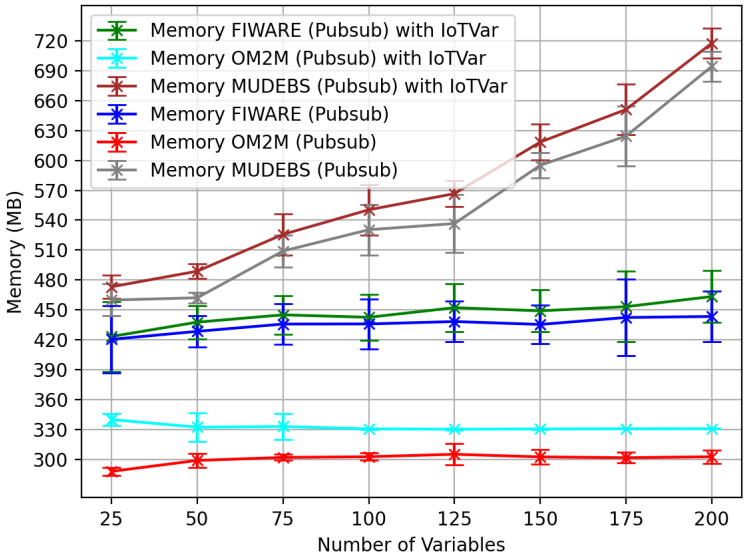
4.3.3 Results for the publish-subscribe interaction pattern

Figure 4.7a shows that CPU usage with IoTvar is higher than without using IoTvar. The overhead induced by the middleware comes from the validations done inside the code to ensure the correct update of variables inside the code, as well as handling multiple potential error cases. In addition, because of the number of proxy objects created and error objects that are created, memory consumption is shown in Figure 4.7b also presented an increase in the memory used by IoTvar. As a consequence, Figures 4.8a and 4.8b also display an overhead in energy consumption for all the three IoT platforms. The differences between using and not using IoTvar are displayed, in percentage, in Table 4.8.

Concerning the FIWARE platform, Table 4.9 shows the results of the U-test with respect to CPU, memory, and energy. For CPU, all the values rejected the null hypothesis, thus concluding that for CPU, the usage of IoTvar does have an impact when used. The memory usage shows a similar behavior, but

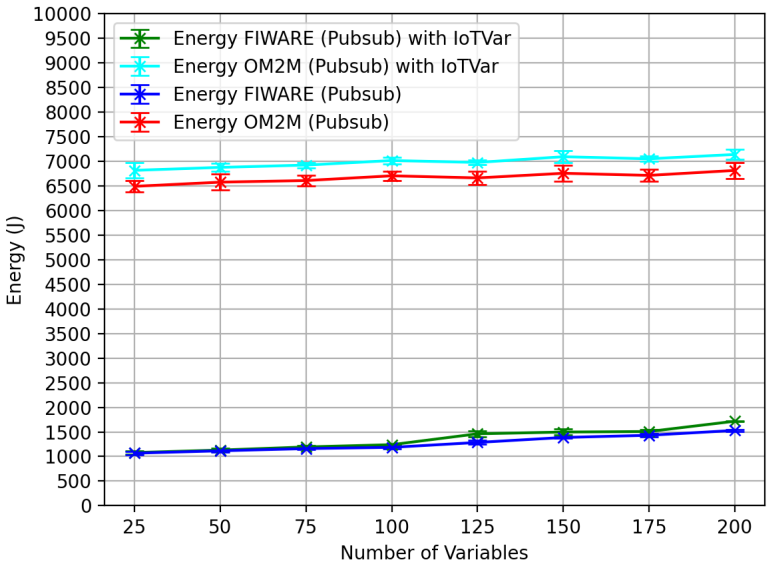


(a) Publish/Subscribe CPU consumption

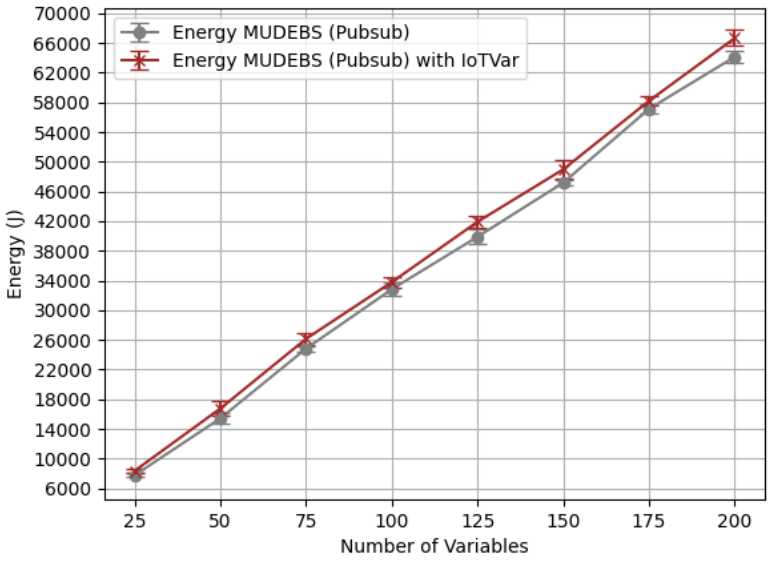


(b) Publish/Subscribe memory consumption

Figure 4.7. IoT platforms pub/sub performance evaluation with and without IoTvar



(a) Publish/Subscribe energy consumption (Orion and OM2M)



(b) Publish/Subscribe energy consumption (muDEBS)

Figure 4.8. IoT platforms pub/sub performance evaluation with and without IoTvar

Table 4.8. Difference between using and not IoTvar for the publish-subscribe interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU FIWARE	4.39%	8.19%	4.94%	7.23%	9.55%	13.99%	10.93%	13.45%
Memory FIWARE	0.68%	2.13%	2.1%	1.52%	3.12%	3.08%	2.43%	4.41%
Energy FIWARE	1.15%	1.36%	2.82%	4.5%	12.81%	7.65%	5.23%	11.51%
CPU OM2M	10.96%	10.72%	16.4%	12.73%	8.37%	4.11%	9.87%	5.92%
Memory OM2M	16.54%	10.56%	9.76%	8.82%	7.92%	8.86%	9.18%	8.91%
Energy OM2M	4.89%	4.46%	4.64%	4.53%	4.61%	4.86%	4.89%	4.65%
CPU muDEBS	7.42%	6.99%	9.28%	4.29%	4.16%	3.56%	3.95%	2.66%
Memory muDEBS	2.86%	5.61%	3.22%	3.7%	5.44%	3.85%	4.18%	3.31%
Energy muDEBS	7.46%	8.21%	5.18%	2.88%	4.93%	3.74%	1.87%	3.86%

with 75 variables the tests show that there is no impact as it has a $p-$value of 0.0909. Similarly, energy consumption shows that IoTvar increases the energy consumption in some of the usages, however, it is also important to highlight that with 25 and 50 declared variables we have a $p-$value greater than 0.05. The results show that there is no significant impact in the tests for this case. Furthermore, the overhead caused by the usage of IoTvar is shown in Table 4.10 with the results from the $A-$index test. In total, there are 2 cases with negligible impact, 5 cases with small impact, 5 cases with medium impact, and 12 cases with a large impact.

Table 4.9. p-values from the U-test for the FIWARE pub/sub interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	0.0119	4.05e-07	0.000467	0.00999	0.000175	7.2e-09	4.03e-09	2.36e-08
Memory	0.259	0.0197	0.0909	0.282	0.0498	0.0211	0.129	0.0371
Energy	0.118	0.0706	0.00751	1.78e-05	3.32e-07	1.24e-07	5.58e-08	4.59e-08

Concerning the OM2M platform, Table 4.11 shows the results of the U-test with respect to CPU, memory, and energy. For all the tests, the U-test showed $p-$values lower than 0.05, rejecting the null hypothesis and thus showing that using IoTvar has an impact. Furthermore, the overhead size caused by the usage of IoTvar is shown in Table 4.12 with the results from the $A-$index test. In total, there were no cases with negligible impact, 1 case with small impact, no cases with medium impact, and 23 cases with a large impact.

Table 4.10. Magnitude from the U-test for the FIWARE pub/sub interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	medium	large	large	medium	large	large	large	large
MEM	negligible	medium	small	negligible	small	medium	small	medium
Energy	small	small	large	large	large	large	large	large

Table 4.11. p-values from the U-test for the OM2M pub/sub interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	7.01e-12	7.73e-10	1.51e-11	3.25e-11	3.61e-08	0.0216	1.51e-08	3.13e-09
Memory	7.01e-12	6.72e-10	1.51e-11	3.25e-11	2.32e-11	8.5e-12	7.35e-12	1.73e-10
Energy	1.6e-09	1.34e-08	2.32e-11	2.01e-10	1.03e-09	1.27e-10	2.24e-10	7.44e-09

Table 4.12. Magnitude from the U-test for the OM2M pub/sub interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	large	large	large	large	large	small	large	large
MEM	large	large	large	large	large	large	large	large
Energy	large	large	large	large	large	large	large	large

Concerning muDEBS platform, Table 4.13 shows the results of the U-test with respect to CPU, memory, and energy. Differently from the other platforms, the CPU tests show a statistically significant difference only for 50, 75, and 200 variables. Similarly, for memory consumption, the difference is significant only for 50 variables, while the other numbers of variables do not show $p-$values over 0.05. The energy consumption shows that the null hypothesis is only accepted for 50, 100, and 175 variables, indicating that for these values there is an impact caused by IoTvar that was statistically noticeable. Furthermore, the overhead size caused by the usage of IoTvar is shown in Table 4.14 with the results from the $A-$index test. In total, there are no cases with negligible impact, no cases with small impact, 3 cases with medium impact, and 21 cases with a large impact.

Table 4.13. p-values from the U-test for the MUDEBS pub/sub interaction pattern.

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	0.0981	0.0404	0.0404	0.0952	0.191	0.0606	0.191	0.0383
Memory	0.0715	0.0404	0.331	0.331	0.0952	0.0952	0.191	0.0952
Energy	0.0101	0.0952	0.0404	0.134	0.0404	0.0404	0.0952	0.0404

Table 4.14. Magnitude from the U-test for the MUDEBS pub/sub interaction pattern.

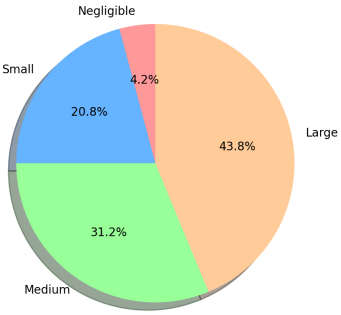
	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	medium	large	large	large	large	large	large	large
MEM	large	large	medium	medium	large	large	large	large
Energy	large	large	large	large	large	large	large	large

4.3.4 Overview of the results

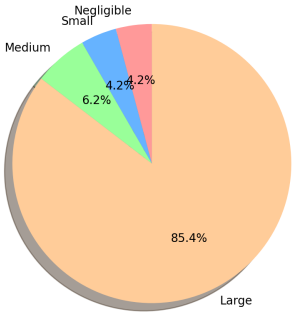
The tests and analysis for the IoTvar middleware show how it behaves in some scenarios over different interaction patterns. The overall results from the tests are shown in Figures 4.9a, 4.9b and 4.9c with the percentage of the magnitude for each tested IoT platform. From these values, we check that IoTvar puts in most cases a bigger load when abstracting the IoT platform. Table 4.15 gives a view over the difference in using and not using IoTVar in percentage showing the overhead caused by IoTvar in percentage.

Table 4.15. Difference in percentage between using and not IoTvar for the supported platforms.

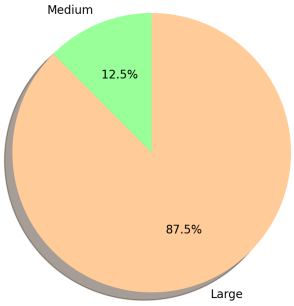
Metric	FIWARE SYNC	FIWARE Pub/Sub	OM2M SYNC	OM2M Pub/Sub	muDEBS Pub/Sub
CPU	5.45%	9.08%	23.09%	9.88%	5.29%
Memory	4.47%	2.43%	4.03%	10.07%	4.02%
Energy	4.88%	5.88%	4.99%	4.69%	4.77%
Global	4.93%	5.8%	10.7%	8.22%	4.69%



(a) FIWARE results



(b) OM2M results



(c) muDEBS results

Figure 4.9. Overall results of the IoTvar middleware

4.4 Conclusion

The heterogeneity of the IoT platforms presents a significant challenge to finding, selecting, and using IoT resources, e.g., devices, sensors, services, and context data. Therefore, it is important to provide techniques that enable clients to easily discover, retrieve and use data produced by them. Due to the different types of data provided by IoT platforms and the various ways to interact with them, it is valuable to be able to gather those data at a low development cost.

This chapter has presented the IoTvar middleware, which provides application developers with a way of interacting with an IoT platform using a few lines of code. For this purpose, IoTvar encompasses proxies representing IoT platform virtual entities. These proxies handle the complexity of interacting with the IoT platform in both synchronous and publish-subscribe interaction patterns. Additionally, IoTvar offers a bypass for the need of understanding the underlying IoT platform-specific API and data model. We have described not only the integration of IoTvar with the FIWARE, OM2M, and muDEBS IoT platforms but also detailed its architecture and how it can be expanded for other platforms.

An evaluation of IoTvar has been performed with all the three supported platforms addressing the balance between the relative cost of IoTvar for both synchronous and publish-subscribe handling of information as well as the benefits for developers. Concerning the cost, the results of the statistical analysis revealed that there is an impact for most of the cases when it comes to CPU, memory, and energy consumption. Furthermore, the results show a global difference of around 5% of increase when using IoTvar. However, these values, although significant, could be acceptable depending on the application characteristics and objectives.

Chapter 5

Impact of Interaction Patterns and IoT protocols on energy consumption

This chapter presents the second contribution of this Ph.D. that is intended to respond to research question 3 (RQ3) introduced in Section 1.2: What is the impact on the energy consumption of widely used protocols by IoT applications and with different interaction patterns? A way of answering this research question is to measure the energy consumption costs induced by the interactions between an IoT consumer application and an IoT platform. As it has been shown in Section 3.2, there is a lack of experiments for isolating the energy consumption of HTTP and MQTT on consumer applications. To fill this gap, we have realized some energy measurements on IoT consumer applications connected to the Internet with a WiFi (802.11n) interface. This application uses the MQTT and HTTP protocols, which are the most common protocols between IoT consumer applications and IoT platforms (as discussed

in Section 3.2). With the results of those experiments, we aim at giving some guidelines for consumer applications and IoT middleware designers.

The chapter is structured as follows. Section 5.1 explains the experimental methodology used to gather energy consumption data from IoT applications. Section 5.2 shows the results gathered and explains, in detail, what can be learned from them to be used in an IoT middleware. Section 5.3 concludes the chapter.

5.1 Experimental methodology

This section presents the methodology used in the experiments for measuring the energy consumption of interactions in IoT consumer applications. Section 5.1.1 presents the experimental conditions in terms of computer, network, energy measurement tool, software, and algorithms. Section 5.1.2 shows the process that allows isolating the energy consumption of the communication part. Section 5.1.3 presents the plan for the experiments with details on the variables that influence each test. Section 5.1.4 discusses the threats that may affect the validity of the experimentation.

5.1.1 Experimental setup

Computers and network

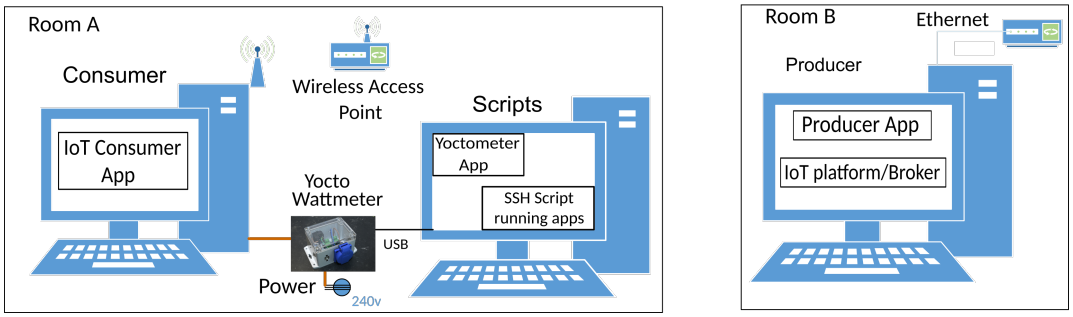


Figure 5.1. Experimental setup

As shown in Figure 5.1, three computers were used to perform the experiments.

1. The **Yocto Wattmeter**, the same wattmeter as shown in Section 4.3, is located between the consumer computer power cable and the wall power

outlet. The Yocto-wattmeter is connected to the energy measurement computer via a USB cable. It uses the Yocto software API to read energy consumption measures and reset the current counter when needed.

2. The **Consumer Computer** used for running the consumer application. The Consumer Computer is connected to the network through a Wifi interface.
   The characteristics of this computer are the following: Dell Latitude E6320 v:01 with 5.68GiB of RAM, a Broadcom (BCM4313 802.11bgn) Wireless Network Adapter driver, and Ubuntu 20.10 Operating system. Furthermore, the battery was fully charged and the computer was always plugged into the electricity.

3. The **Producer Computer** is used for simulating an IoT platform. It runs a process that produces data. It is a fixed computer connected to the Internet through an Ethernet interface.

4. The **Script Computer** was used (i) for running the scripts responsible for starting all applications on the client and server computers, and (ii) for reading the energy consumption measures on the Wattmeter.

For **MQTT**, we use the *Mosquitto* broker version 3. The producer and the consumer were developed using the open-source Eclipse *Paho* library for Java. For **HTTP Request/Reply**, the consumer uses HTTP/1.1 with the *java.net.http.HttpClient* Java library. For **HTTP Publish/Subscribe** we also use HTTP/1.1 and the consumer includes an Undertow Server to receive HTTP publications. We highlight that in a real scenario, the consumer application does not choose the version of the HTTP protocol used by the server nor does the configuration of the server concerning connection management. In this context, the usage of HTTP/1.1 is widely supported by servers and clients whereas other versions such as HTTP/2 are still less common.

Algorithms

We present first the algorithms on the consumer application side for the two interaction patterns, request/reply (Algorithm 1) and publish/subscribe (Algorithm 2), and then the algorithm on the producer (Algorithm 3) that simulates an IoT platform. For the request/reply algorithm, the *period* input is equivalent to the refresh time, the period between two requests. On the other hand, for the publish/subscribe pattern, this rate is controlled by the producer and

its *period* of publications. Moreover, the *payload* size is the second input of the producer algorithm.

Algorithm 1: Consumer Request/Reply

```
Main(period)
begin
    producer ← httpInitialisation(URI)
    while true do
        value ← producer.getValue()
        sleep(period)
    end
end
```

Algorithm 2: Consumer publish/subscribe

```
Main(void)
begin
    server ← initializeServer(URI, handler)
end
handler(receiver)
begin
    value ← receiver.getValue()
end
```

Algorithm 3: Producer

```
Main(period, payload)
begin
    dest=initializeServer(URI)
    while true do
        dest.send(payload)
        sleep(period)
    end
end
```

Finally, a script runs on the measuring computer. It takes as input the *period* between two publications, the *payload* size, and the *duration* of the

experiment. The script starts the consumer and the producer, then sleeps for one minute for initialization and consumer warmup purposes. Then, the energy meter on the wattmeter is reset and is ready to start gathering the energy measure at the end of the experiment. Finally, the script reads the consumed energy on the consumer application from the wattmeter and stops the producer and the consumer.

5.1.2 Process to isolate the communication energy consumption

Using a wattmeter has the following disadvantage: there is no isolation of the application or any particular process in the measurement, as the wattmeter measures the energy consumption of the computer as a whole. For proper measurement of the impact of an application, it is necessary to make two measures: (1) the measure of the energy consumption without the application, and (2) the measure of the energy consumption with the application. In Figure 5.2, we present the following measures of energy consumption of the consumer computer:

- $M_{idle+jvm}$: we start the Consumer computer with the consumer application but without any interaction with the producer application (blue + orange on Figure 5.2)

- $M_{idle+jvm+interactions}$: we start the Consumer computer with the full consumer application (blue + orange +green on Figure 5.2)

The results that are presented in Section 5.2 only show the interaction cost (the upper part in green on Figure 5.2). We obtain this value with this formula: $M_{idle+jvm+interactions} - M_{idle+jvm}$. As a consequence, the standard deviation of the result is the addition of the standard deviation of the two measures.

5.1.3 Experimental plan

Table 5.1 presents the combinations of interaction patterns and protocols for which we have handled 5 families of experiments. We measure: (1) the impact of the interaction pattern through families F1 and F2; (2) the impact of the protocol with families F2 and F3, and (3) the impact of the QoS for MQTT with families F3, F4, and F5.

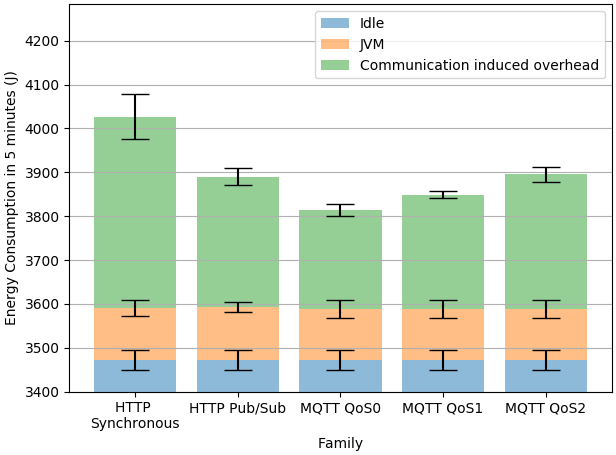


Figure 5.2. Energy consumption measures

Family	Interaction pattern	Protocol
F1	Synchronous	HTTP
F2	Publish/Subscribe	HTTP
F3	Publish/Subscribe	MQTT QoS0
F4	Publish/Subscribe	MQTT QoS1
F5	Publish/Subscribe	MQTT QoS2

Table 5.1. Families of experiments

The message rates used in the experiments were 1, 2, 4, 8, and 16 messages by second. The tests at 32, 64, and 128 messages per second with both interaction patterns using HTTP started to receive a significantly lower amount of messages, as a consequence, we did not keep those results.

The payload used in the experiments were 24, 48, 240, 1320, and 1560 bytes. We start with 24 bytes since it is assumed that this payload is about the usual value for an IoT payload. The last two values were chosen considering the MTU, which was measured at 1500 bytes for our experiments. One lower than this value and the other higher for comparison purposes on the energy-consumption influence of such a scenario.

We run 125 experiments: 5 families of experiments (see Table 5.1)* 5 message rates * 5 payloads. For each experiment, we compute the mean and standard deviation among 30 measures. Three more measures were done with the consumer application also running Wireshark to analyze the obtained results if necessary. As the usage of Wireshark increases the energy consumption

of the machine, we do not include those tests for computing the mean and the standard deviation. In total, we performed 33*125= 4 125 tests.

Each measuring test had a total duration of 8 minutes. This was organized with one minute for warm-up, where the producer started the message exchange with the consumer. This part is followed by a measurement of the energy consumption for 5 minutes while the producer was exchanging data with the consumer. Finally, two more minutes of sleep time to reset the experiment, and slow down the computer and the network conditions before starting the following test. 4 125 tests of duration 8 minutes necessitate around one full month of experiments. Additionally, for $M_{idle+jvm}$, 60 tests were performed, and we double the number of tests to obtain low standard deviation and confidence intervals.

5.1.4 Threats to validity

We present below potential threats to the validity of our study and how we propose to minimize their effects.

Computer conditions: The activity of the computer cannot be fully controlled. As a consequence, we report some discrepancies in the measured values. To reduce these discrepancies, we have shut down or disabled all unnecessary processes of the operating system as well as using the lowest brightness and connecting to the device via ssh to reduce user tampering. To minimize the standard deviation and obtain a more consistent result, each of the experiments was run a total of 30 times.

Network conditions: The conditions of the network while doing the tests were optimal. The gathered data showed that there was no packet loss during the tests and the latency remained low and stable at around 23ms.

Temperature at which the experiments are conducted: during the initial experiments, the climate did not rise above 25 degrees Celsius. However, on some days when the external temperature rose between 28 and 32 degrees, the fluctuations in energy consumption increased. These fluctuations may be due to the need for the equipment cooling systems to increase their output to keep the components of the equipment in the correct temperature conditions. To address this threat, the client's computer was moved to an air-conditioned room where the computer was always at a cold temperature. This resulted in a reduction of the standard deviations of the measurements, making the results more stable. The experiments realized in the air-conditioned room were for the message rate of 8m/s and the payloads of 1320B, 1560B, and 3120B.

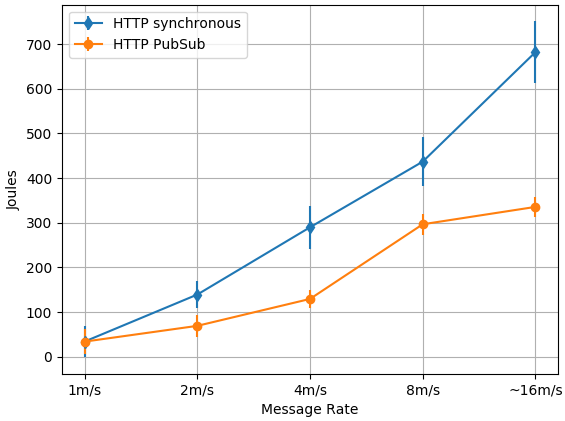


Figure 5.3. Energy consumption 24B, Interaction Pattern Comparison

5.2 Results

This section shows the results of the experiments. Section 5.2.1 gives details on how the interaction pattern can affect energy consumption. Section 5.2.2 shows how the choice of the protocol for the application can affect energy consumption. Section 5.2.3 is focused on the MQTT protocol and shows what are the differences in energy consumption when considering different QoS for it. Section 5.2.5 presents guidelines dedicated to developers of IoT consumer applications considering the results shown.

5.2.1 Impact of the interaction pattern

For a fair comparison of the interaction patterns, we compared only the results obtained with the HTTP protocol for which we measured the two interaction patterns.

Figure 5.3 presents the results of the energy consumption for a 24Bytes payload for both interaction patterns. Table 5.2 presents, in percentage, the synchronous pattern overhead over the publish/subscribe pattern. This is a synthesis of all the realized measures (all the message rates).

The results of the experiments show that with the same number of received observations, the synchronous pattern consumes around 92% (mean of all the message rates and payloads results) more energy than the publish/subscribe interaction pattern, being almost two times less efficient. This happens as the client needs to process the request for the server and wait for a reply whereas

Payload	Overhead in %
24B	+94.03%
48B	+89.96%
240B	+106.90%
1320B	+85.16%
1560B	+85.50%
Mean	+92.31%

Table 5.2. Synchronous pattern average overhead over the publish/subscribe pattern

in pub/sub it will only need to wait for notifications from the server.

5.2.2 Impact of the application protocol

For the comparison of the protocols, it was desired to conduct a fair comparison of the two protocols, comparing the MQTT QoS 0 and HTTP Pub/Sub as they both propose an "at most once" semantics. As observed in Figure 5.4 and Table 5.3, MQTT outperforms HTTP in terms of energy consumption and the number of bytes by Joule.

We observe that in terms of energy, the MQTT protocol outperforms HTTP by 20% on average while having the same interaction pattern and the same semantics. This happens because of the purpose of each protocol. While HTTP has more processing on top of the data received by the client, as it needs to look into further validations (e.g size variable header, parameters, etc), MQTT is proposed with a more lightweight structure that, for example, has fixed headers, enabling less intensive processing by the client.

Payload	HTTP vs MQTT in %
24B	+28.07%
48B	+23.40%
240B	+31.05%
1320B	+2.58%
1560B	+18.63%
Mean	+20.75%

Table 5.3. HTTP vs MQTT average overhead with all the message rates

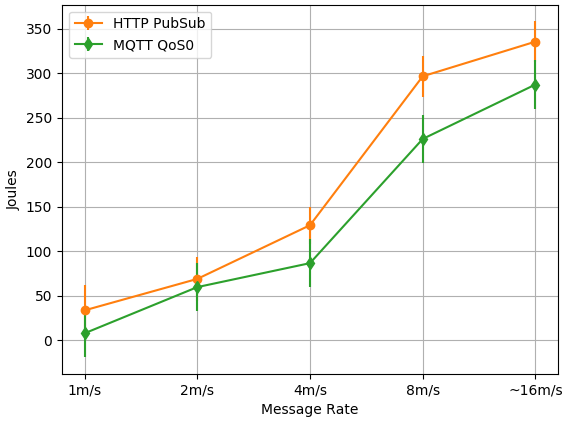


Figure 5.4. Energy consumption for a 24B payload, Protocol Comparison

5.2.3 Impact of the QoS in MQTT

In Figure 5.5, we compare the measures of energy consumption for the three MQTT QoS with the 24B payload. Table 5.4 presents a synthesis of the overheads for all the payloads.

Payload	QoS1/QoS0	QoS2/QoS1	QoS2/QoS0
24B	+24.64%	+46.58%	+79.72%
48B	+17.76%	+51.09%	+78.20%
240B	+58.67%	+27.08%	+104.46%
1320B	+12.16%	+88.10%	+111.76%
1560B	+21.83%	+53.31%	+86.45%
Mean	+27.01%	+59.77 %	+92.12%

Table 5.4. MQTT QoS overheads

Taking into consideration all the measures realized, meaning all the message rates and payloads, the comparison of QoS shows that QoS 0 consumes around 27% less energy than QoS 1 and 92% compared to QoS 2 with the same number of received observations. QoS 2 consumes 60% more energy than QoS 1. Having similar results in the comparisons of QoS 0, QoS 1, and QoS 2, to what was observed in the related work [Told19]. The difference is that we can measure the consumer side only while they use the same device as producer and consumer and as a result, can not differentiate consumer from producer energy consumption.

A deeper look at the results shows that the impact of the QoS using MQTT is related to the number of messages exchanged during the experiment. QoS

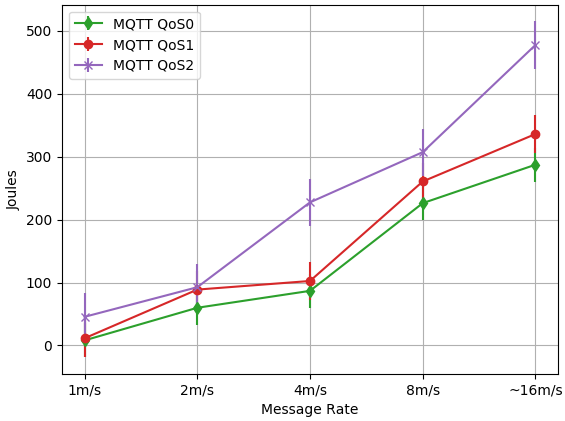


Figure 5.5. Energy consumption for a 24B payload, QoS Comparison

0 had the smallest amount of messages exchanged between the broker and the consumer because of the fire-and-forget mechanism (Sending messages and not verifying the arrival) it implements and resulting in the lowest energy consumption. MQTT QoS 1 followed a similar path but as it increased the number of messages, due to the acknowledgments of the client, it resulted in bigger energy consumption when compared to MQTT QoS 0. Finally, QoS 2 with its bigger amount of messages exchanged between the broker and the client doubled the number of packets exchanged and ended up almost doubling the amount of energy used.

5.2.4 Impact of the payload

Figure 5.6 presents the bytes/Joule for different payloads for the fixed rate of 8 messages per second for the 5 families of experiments.

The usage of a payload up to 3120 bytes presented a moderate increase in the experiments (mean 9%), having cases with even lower consumption for HTTP. The fragmentation of the messages according to the MTU (1500 bytes) does not seem to have a relevant impact on energy consumption. The experiment impacted the most by the increase was MQTT QoS 0, having up to 21.89% more energy consumption. Concerning HTTP publish-subscribe and request-reply, both seemed unaffected by the changes in the payload as the payload of 24B was slightly higher than the one with 3120B (Table 5.5), which had the message broken into 3 fragments according to the size of the MTU. The behavior of the payload seen in HTTP is further confirmed when checking

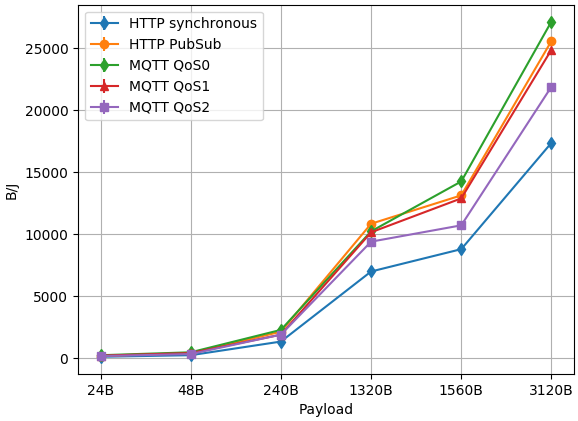


Figure 5.6. Bytes Received by Joule Payload Comparison

Family of experiment	Overhead in %
HTTP synchronous	-1.36%
HTTP Pub/sub	-1.32%
MQTT QoS0	+21.89%
MQTT QoS1	+15.45%
MQTT QoS2	+11.24%
Mean	+9,18%

Table 5.5. Payload overhead from 24Bytes to 3120Bytes

the number of TCP connections with 24 bytes and 3120 bytes which remained the same. Furthermore, MQTT is more impacted by the payload because the protocol created only one TCP connection through all the test phases and exchanged messages in this established connection. This causes the payload to become a bigger part of the energy consumption considering that MQTT has a 2 bytes fixed header, while the HTTP header does not have a limit (A limit can be set by the server), but in the case of the tests done, it is around 106 Bytes. Besides those differences by family of experiments, the lesson of this comparison is that the number of Bytes by Joule is augmented significantly for all the families while augmenting the payload. An explanation for this result comes from the cost of the software call stack necessary to handle one message whatever the size of the message is. To better verify this, a possibility is to increase the amount of data exchanged by bigger amounts and compare it with

energy measurements from lower amounts of data.

5.2.5 Guidelines for IoT consumer application designers

From the results of the experiments, we provide in this section guidelines for IoT consumer application designers to reduce energy consumption at the end user device side.

Group several observations in one message

We have shown that using different payloads, from 24 up to 3120 bytes, has a small impact on the energy consumption of the application. If the application needs multiple sensor observations, we advise combining the different observations into one single message. Some IoT platforms, such as FIWARE/Orion, provide the possibility to query (or subscribe to) a group of sensors. This possibility has clearly to be chosen by application developers. Furthermore, the usage of such a strategy at the middleware level can lower the amount of consumption for IoT applications that use it, while still lowering the cost of development of the application because of the usage of middleware to handle communication with IoT platforms.

Favor the Publish-Subscribe interaction pattern

The comparison of interaction patterns showed that for the same frequency of requests and notifications, the publish/subscribe pattern consumed on average 92% less energy than the request/reply pattern. As a consequence, we advise favoring the publish/subscribe pattern.

We have to mention that this advice may depend on the IoT application and the IoT platform. If the frequency of requests is far lower than the frequency of publications, the synchronous pattern can be an option because the client can better control the number of messages being exchanged. This happens due to IoT platforms lacking features of filtering information, thus increasing the amount of data sent to the applications. This highlights that the filtering of information is an important feature of IoT platforms for both synchronous and publish/subscribe patterns.

Favor the MQTT protocol over the HTTP protocol

For the publish/subscribe pattern, the comparison of the MQTT and HTTP protocols shows that MQTT has 20% less energy overhead in comparison to

HTTP. The advice is then to favor the MQTT protocol for the publish/subscribe pattern. For IoT platforms, this information highlights the importance of supporting not only multiple protocols but also providing the applications with different choices of interaction patterns.

Choose the QoS appropriate for your application

With the MQTT protocol, if your IoT application supports losing some observations, prefer QoS 0 since it involves less energy consumption. If the application cannot afford to lose observations, use QoS 1 instead of QoS 0 as it provides a complementary service to TCP's reliability by ensuring that each message is received at least once. Keep the QoS2 for exactly-once semantic requirements as it presents an overhead of 92% and should be used in conditions that require no duplication of messages.

Examples of benefits from those guidelines

The benefits of those guidelines to develop IoT applications can be better seen when viewing with a bigger space of time. As an example, an IoT application running for one year using HTTP pub/sub sending 4 messages per second with a payload of 24B will consume around 31,01 MegaJoules while another application running with the same parameters but using MQTT QoS 0 will consume around 8,40 MegaJoules. Furthermore, if the messages are grouped into a single message, and sent once per second, we can achieve consumption of around 4,20 MegaJoules for both HTTP pub/sub and MQTT QoS 0 with a payload of 24 Bytes. As an example, for a regular notebook battery with around 360 KiloJoules, an IoT application using HTTP or MQTT and grouping messages could lead to a lifetime of around 31 hours, on the other hand, without grouping and using HTTP synchronous we have around 9 hours of battery (71% less).

5.3 Conclusion

Energy consumption is a first-class concern in the development of future IoT applications. As the number of devices and the number of applications related to IoT will keep growing in the near future, there is a new requirement for the developers and the users to regulate and improve the energy consumption of IoT applications not only on the connected object side but also on the consumer application side.

The energy consumption of IoT consumer applications was measured on user devices connected with WiFi (802.11n). The impact on energy consumption was shown for different interaction choices and is summarized as follows. (i) The results show that for the same amount of received observations, the publish/subscribe interaction pattern has lower energy consumption (around 92% lower) than the synchronous interaction pattern; (ii) For the publish/-subscribe interaction pattern, MQTT consumes less than the HTTP protocol (around 20% less). (iii) The payload has a low impact on energy consumption having a 9% overhead from 24 to 3120 bytes payloads. From these results, guidelines for IoT consumer application designers were elucidated, for example, developers should favor the publish/subscribe pattern and group several observations in one message when possible.

The results shown are also important when it comes to IoT middleware, as the guidelines can serve as a base for the design of IoT middleware for IoT consumer applications. Implementing those strategies at the middleware level may have a strong impact on reducing IoT application energy consumption while keeping a low development effort.

Chapter 6

Energy-efficiency/Awareness in IoTVar

This chapter presents how IoTvar has been extended to handle energy efficiency and energy awareness. This proposal is an answer to RQ4 introduced in Section 1: What are the strategies to be proposed by an IoT middleware to reduce the energy consumption of IoT consumer applications? The objective is to validate the usage of the strategies at the middleware level and evaluate the positive impact in terms of energy consumption.

The chapter is structured as follows. Section 6.1 shows guidelines that can be followed when introducing energy-efficient mechanisms in a middleware. Section 6.2 presents the architecture of the IoTvar middleware with the

necessary modifications to support energy-efficient mechanisms. Section 6.3 describes all the energy-efficient mechanisms implemented in the middleware along with how it can be extended with an energy budget paradigm to provide energy-awareness mechanisms for the end user. Section 6.4 presents the results and evaluation of the IoTvar middleware compared with its usage without energy-efficient/aware mechanisms. Section 6.5 presents a discussion over the results and limitations of the evaluation. Section 6.6 provides concluding remarks.

6.1 Guidelines for energy-efficiency and energy-awareness in IoT middleware

IoT middleware is a proper software level for improving energy efficiency as it is the place where energy-efficient interaction practices may be shared by many applications. Improving the energy efficiency of IoT middleware may be achieved through the introduction of energy-efficient strategies. Although many strategies have been proposed by IoT middleware as shown in Section 3.1, none of them were dedicated to IoT middleware for IoT consumer applications. In order to overcome this limitation, we propose below a set of guidelines to aid designers of IoT middleware in providing energy efficiency to IoT consumer applications: *(1)* Switch of communication protocols, *(2)* Message grouping, *(3)* Refresh-time adaptation, *(4)* Interaction pattern switch, *(5)* Energy budget management.

6.1.1 Switch of communication protocols

IoT middleware for consumer applications may abstract several protocols, and thus choose the protocol in function of the IoT platform-supported protocols. One feature that can be implemented at the middleware level is the switch of communication protocols for the IoT application. For example, an IoT application that wants to communicate with an IoT platform, using the Pub/Sub pattern which is supported by both MQTT and HTTP, could lead to the middleware automatically choosing the MQTT because it is known that is a more efficient protocol when it comes to energy consumption.

6.1.2 Message grouping

As shown in Section 5.2.4, the middleware can use message grouping to lower the amount of communication and thus lower the amount of energy consumption. This is possible due to the decrease in the number of messages to be sent in a space of time while increasing the payload sent in each request.

6.1.3 Refresh Time Adaptation

The update of data needs to be done regularly by the middleware, it can be done in the publish/subscribe interaction pattern if there is such a mechanism in the IoT platform to limit the minimum time between two notifications. It can also be done in the request/reply mode, where there is a request sent periodically by the middleware. In either case, this refresh time of the data needs to be set, and the lower it is, the more energy the middleware will use because the cost of communication increases as more requests are done in a smaller space of time. To overcome this issue, carefully managing the refresh time of data brings further energy efficiency. This management can be done by the middleware to take into account a maximum energy budget set by the user and then augment the refresh time if necessary with the objective to lower the energy consumption.

6.1.4 Interaction pattern switch

IoT middleware can provide different types of interactions with IoT platforms or other components of an IoT system. One example of interaction is the pub/-sub where the IoT middleware waits for notifications with data to be sent to it. Another is the request/reply interaction in which the middleware acts by starting the communication to retrieve data. When it comes to energy efficiency, the usage of pub/sub can be more efficient than request/reply by itself, but when also applying a grouping strategy, the energy consumption of the request/reply could be less than pub/sub. With this information, the IoT middleware can provide the interaction switching strategy that makes use of energy consumption data from the middleware to change the type of interaction, if possible, to use the one at the time that uses the least. As an example, the IoT middleware could use an energy model to determine if the energy consumed by using the request/reply with grouping could be switched with a pub/sub interaction pattern for lower energy consumption. Furthermore, depending on the support for the pub/sub by, for example, an IoT platform, there could be a possibility

of grouping with pub/sub, which would be more efficient than using grouping with request/reply.

6.1.5 Energy Budget management

The management of the above strategies could be driven by the middleware to respect an energy budget that could be defined by the end user. For this purpose, the middleware should provide energy awareness through the estimation of how much energy is consumed by the interactions. According to the estimation, energy-efficient interaction strategies will be chosen to reduce, if necessary, energy consumption at the middleware level.

6.2 IoTvar Energy-efficient/Aware architecture

The architecture of IoTvar, presented in Chapter 4, enables the integration of an energy-efficient/aware component to be added without heavy modifications of the middleware. Figure 6.1 shows the proposed modified architecture of IoTvar with the new components added to support energy-efficiency/awareness. The new components are highlighted in dark green color.

To provide energy efficiency, energy awareness, and energy budget management, a new layer for the IoTvar middleware has been added: the energy layer. The *Generic Energy efficient/aware* component created at this layer will be detailed in Section 6.3. In addition, each specific *unmarshaller* has to be specialized to unmarshal a group of variables. The synchronous handler has also to be updated to request several variables at the same time.

An implementation of this architecture has been implemented for the *FIWARE* IoT platform: it handles the *message grouping*, *refresh time adaptation* and *energy budget management* strategies for the *request/reply* interaction pattern.

6.3 Energy-efficient/Aware mechanisms in IoTvar

The mechanisms to reduce the energy consumption in applications that use IoTvar have been designed taking into account the guidelines presented in Section 6.1. The initial schema, which represents the generic component in the new layer of the architecture, is presented in Figure 6.2. First, IoTvar receives input from a configuration file, described in Table 6.1, that contains directives

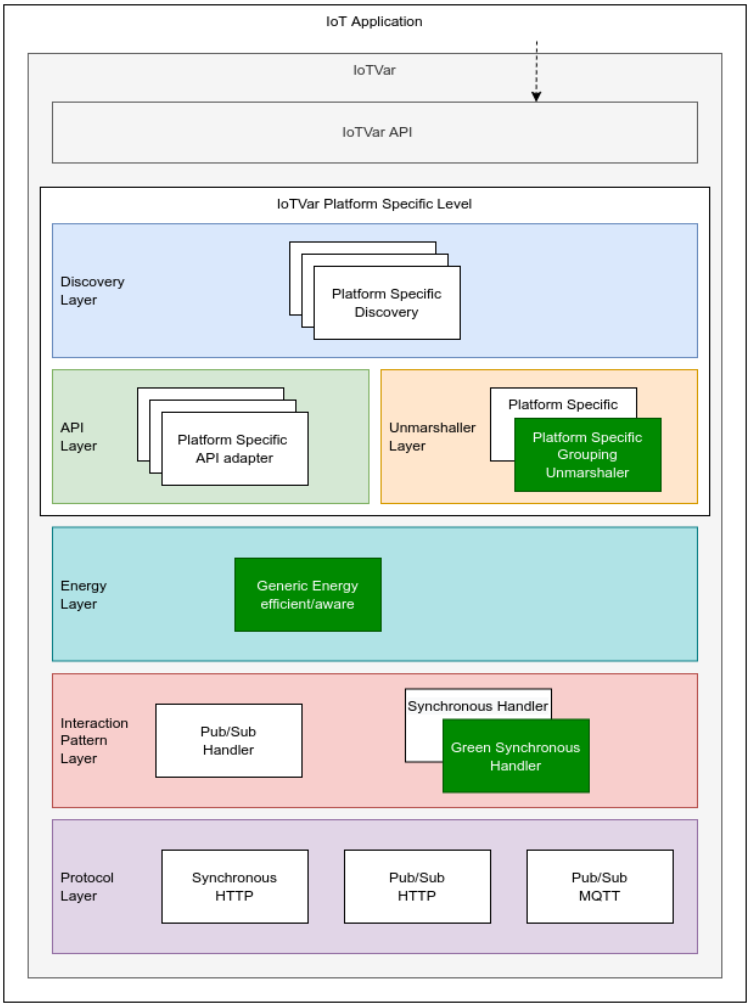


Figure 6.1. IoTvar architecture with energy-efficiency/awareness

about how to behave in certain cases such as when the application has a certain energy budget or to limit certain energy-efficient mechanisms to be able to provide more QoS. This configuration file is used by the *Energy Awareness* module that takes decisions based on *(i)* the number of declared variables, *(ii)* the network status, *(iii)* the energy budget, and *(iiii)* the energy model. This information is used by an algorithm that provides output directives for the energy-efficient module to group messages and control the rate of communication.

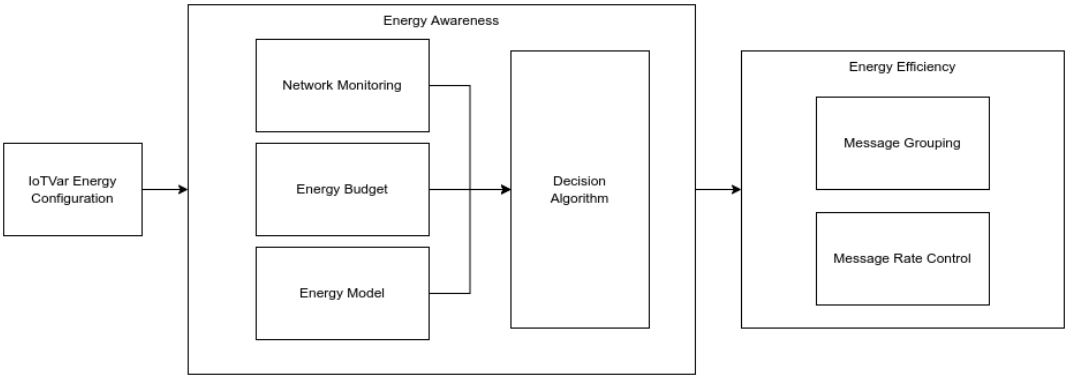


Figure 6.2. Inside the IoTvar energy efficient/aware component

6.3.1 Configuring IoTvar energy efficient mechanism

IoTvar provides a default behavior for variables declared using the energy-efficient handler, but it can also read from a configuration file provided by the user that enables refinement of the behavior. In case the user wants to limit the processing that IoTvar does to gather information for its energy-aware module or how much IoTvar controls the interactions with IoT platforms, the configuration file provides the possible properties that can be passed to the code. The configuration file properties available are shown and explained in Table 6.1.

6.3.2 Network and variable status

When IoTvar variables are declared, it updates the status of the application. This status contains information regarding network conditions and declared variables. The network condition information is data related to package loss, latency, availability of the IoT platform, and type of network being used (e.g. WiFi, Ethernet, 3G,4G,5G). The declared variable information contains data regarding the number of variables, type of handlers used, refresh time, and, if enabled, the energy budget available.

6.3.3 Energy budget

The energy budget is an important aspect of the energy-efficiency mechanism as it will be actively used to lower the energy-consumption if necessary. It is compared to the estimated energy consumption computed with the model every time the status of IoTvar is refreshed which could consequently change

Table 6.1. IoTvar configuration file properties

Property	Behaviour	Default Value
Status Refresh Period (in seconds)	period for recalculating the status of the middleware (network and variable conditions)	10
Maximum Freshness Increase (in seconds)	This configuration imposes a limit to how much IoTvar will increase the refresh time. When IoTvar increases the refresh time the QoS will be lowered, thus this configuration will limit this degradation of QoS up to a maximum (by refresh period).	10
Use Energy Model	This property indicates if IoTvar should use the energy model to calculate the energy being consumed taking into account the variables that are declared in the code. If it is *"false"*, IoTvar does not report the estimation of the energy being used.	false
Use Energy Budget	If the energy model is *"true"* and this also *"true"*, IoTvar will start to take into consideration the budget provided in the property *Energy Budget*.	false
Energy Budget (in Joules for 5 minutes)	This value is used by IoTvar to lower the amount of energy calculated until reaching the desired budget. This value is given in Joules and is the budget for 5 minutes.	400

the refresh time of the grouped variables. If the energy consumption exceeds the budget, IoTvar increases the number of variables being grouped and also increases the refresh time of each variable while decreasing the QoS.

6.3.4 Energy model

The proposed model takes into account the main resources used to update data inside the code. The values for the variables and constants in the model were calculated through tests to find the good values that would give a close estimation of the real energy consumption values returned by a wattmeter. The model takes into consideration the following variables and constants to calculate the energy being consumed:

- nb_G: The number of groups of variables that are internally created by IoTvar; There is one group by refresh time;

- $nb_{V_{G_i}}$: The number of variables inside a given group;
- R_{G_i}: The refresh time of a group in seconds;
- C_V: A Constant energy value for each variable that is inside a group. It represents the cost of the small processing each variable adds inside a group.
- C_{net}: A Constant given for the calculation of the network. It is the estimated energy that is consumed when making a request for a group.
- M_{netS}: A modifier for the network calculation that depends on the latency, package loss, and reachability of the IoT platform;
- M_{netI}: A modifier that depends on the network interface being used (WiFi or Ethernet);
- C_{cpu}: A constant given for the CPU processing of one group of variables in IoTvar.

The total energy consumed in Joules is calculated based on the energy used by IoTvar during 5 minutes and is compared to the energy budget to enable the middleware to increase the refresh time of the groups and thus lower the energy consumption. The following equation represents the energy model to calculate the energy consumption of IoTvar in a period of 5 minutes:

$$\sum_{i=0}^{nb_G} \frac{(C_V * nb_{V_{G_i}}) + (C_{net} * M_{netS} * M_{netI}) + C_{cpu}}{R_{G_i}}$$

Furthermore, the model may also be used for context awareness purposes. The application designer may: *(1)* Show to the end user the estimation of how much energy the application is using at runtime or *(2)* Evaluate at design time the energy consumed by the application.

6.3.5 Energy aware Algorithm for energy efficiency

Algorithm 4 is a representation of the function used by IoTvar to calculate the increase of refresh time of the groupings of variables. This algorithm receives the current refresh time of the groupings inside IoTvar, the energy configuration, and checks if the minimum time, which is retrieved from the configurations, has passed. If the checking is false, then the algorithm simply returns the original refresh time, otherwise, it continues to calculate. The refresh time

increase is dependent if there is reachability to the IoT platform, ping to the platform (latency and package loss), and the network interface in use. After gathering this information, the algorithm will add these values and return a new refresh time for the grouping. Furthermore, if the configuration of IoTvar enables the usage of the energy model and budget, the algorithm will calculate the increase of refresh time, if needed.

Algorithm 4: Energy aware algorithm

```
getNewGroupRefreshTime(currentGroupRefreshTime, config)
begin
    if refreshTimePassed(config.statusRefresh) then
        platformRefreshTime = isPlatformConnected()
        pingRefreshTime = pingToPlatform()
        networkRefreshTime = networkInterfaceInUse()
        newGroupRefreshTime = currentGroupRefreshTime +
         platformRefreshTime + pingRefreshTime +
         networkRefreshTime
        if config.usingEnergyModel and config.usingEnergyBudget
         then
            estimatedEnergyUsed =
             getIoTVarEstimationUsingEnergyModel()
            budgetRefreshTime =
             getRefreshTimeEstimateEnergy(estimatedEnergyUsed,
             config.energyBudget)
            newGroupRefreshTime += budgetRefreshTime
        end
        return newGroupRefreshTime
    end
    return currentGroupRefreshTime;
end
```

6.4 Evaluation

This section reports a quantitative evaluation of the IoTvar middleware using the energy-efficient/aware mechanisms for the FIWARE platform. This evaluation involves a performance assessment that considers the same application written with/without IoTvar (i.e., directly accessing the IoT platform) and with IoTvar but using the energy-efficient mechanisms, aiming at measuring

the impacts of the new mechanisms in the middleware in terms of CPU and energy usage. The application goal is the same as presented in Chapter 4, to receive and display, in the console, the data that is returned from a temperature sensor. The tests done with this application vary the number of sensors from 25 to 200 in a 25 sensors step (25, 50, ..., 200), with a refresh time of 1 second for each sensor, and having a small local history of the 10 latest values for each sensor. The payload of the data of one sensor is around 117 bytes while for 200 variables is around 10282 bytes.

The performance measurements of this evaluation are similar to the tests shown in Chapter 4 for IoTvar. We collect the performance measurements (CPU and energy consumption) by wrapping the method called the energy-efficient synchronous handler. This is implemented using AspectJ, which provides an encapsulation around the main methods of the IoTvar structure and is woven into the IoTvar code.

Mann Whitney's U-test is used for checking if there is a statistically significant difference between the analyzed samples (IoTvar with and without energy-efficient strategies). Following these results, an effect-size statistical analysis is run over the test data using the $A-$index by Delaney and Vargha that are interpreted with Hess and Kromrey magnitude levels (Negligible, Small, Medium, and Large).

6.4.1 Setup

Fig. 6.3 illustrates the setup of the tests. The computer that executes the client application consuming IoT data and the server computer which simulates IoT sensors that send IoT data through the IoT platform are the same used in Section 4.3.1. Furthermore, the client computer is plugged into the same YoctoPuce YoctoWatt wattmeter to collect energy-related measures and the client application communicates with the server through a similar local isolated WiFI network.

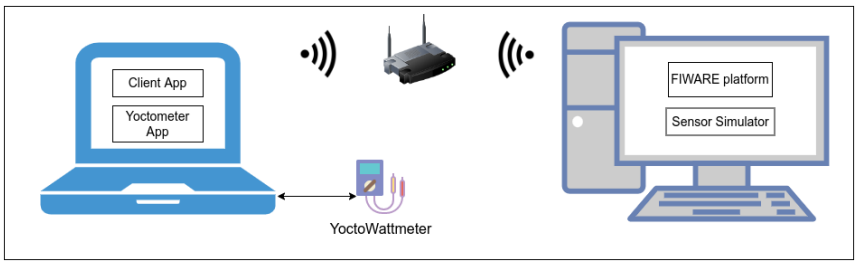


Figure 6.3. IoTvar energy-efficient experimental setup

When the tests are running, the IoT application creates the number of variables for the specific test and starts calling the FIWARE platform for information on the variables. In the former version of IoTvar, for each variable one HTTP call was made. In the new version, using the strategies for energy efficiency the IoTvar middleware is capable of grouping the variables if they have the same refresh time. Thus, the middleware makes one single call by group.

Platform consideration We have to mention that this is possible due to the FIWARE platform having the capability of grouping requests in its API. However, this capability has limits. One example is using the FIWARE platform, which does not support proper filtering for the variables when grouping (i.e. it imposes the same filter for each variable of the group). This implies that FIWARE variables that need filtering cannot be used along with the energy-efficient handler of IoTvar.

6.4.2 Results with the Wattmeter

The data from the tests were gathered and Figures 6.4 and 6.5 show the performance of energy-consumption and CPU usage by IoTvar. The usage of the strategies for energy efficiency has lowered the energy usage of IoTvar, reaching close to 1000 Joules instead of the 1150 Joules using the non-efficient version for 25 variables. Similarly, the CPU usage that reaches 35 seconds when looking at 25 declared variables was lowered to less than one second. This can also be seen in Table 6.2 which shows that for the percentage change of the CPU there is a huge increase of more than 99% for 200 variables, and for the energy consumption, it is around 45% more usage at 200 variables.

Table 6.2. Increase of Joules using IoTvar without energy-efficient strategies

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	97.36%	98.68%	99.04%	99.23%	99.36%	99.46%	99.5%	99.57%
Energy	11.44%	21.62%	24.82%	29.73%	37.17%	40.53%	42.29%	45.87%

Concerning the statistical analysis of the data, Table 6.3 shows the results of the U-test for CPU and energy. For both the CPU and energy the values reject the null hypothesis, thus concluding that not using the energy-efficient/aware strategies for IoT middleware inside IoTvar brings an impact on these resources.

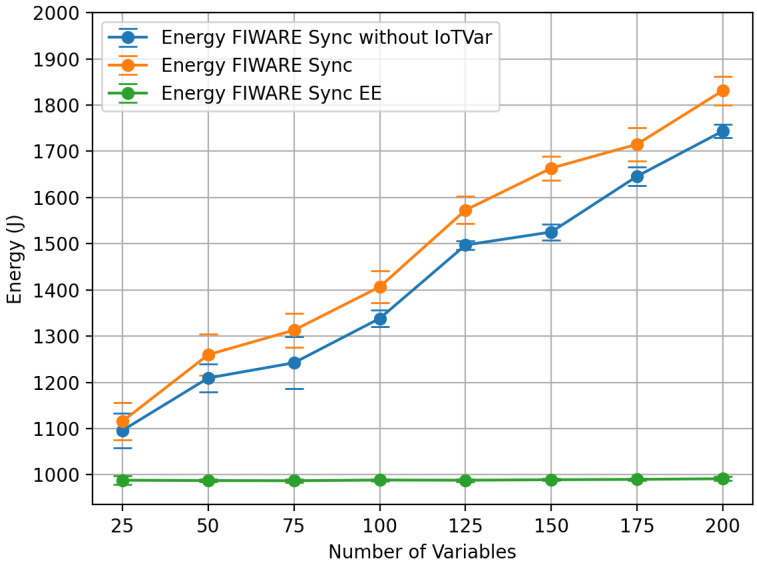


Figure 6.4. IoTvar energy consumption using EE strategies

The $p-$value shown can better be seen in Table 6.4 using the $A-$index test which interprets the values. From it, there is a clear view that the impact of not using energy-efficient strategies causes a large impact on the resource consumption of the IoT application using an IoT middleware such as IoTvar.

Table 6.3. p-values from the U-test for IoTvar and the FIWARE platform

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	2.94e-11	2.94e-11	2.94e-11	2.94e-11	2.94e-11	6.35e-11	1.37e-10	6.5e-09
Energy	2.44e-11	2.17e-11	1.96e-11	1.72e-11	2.02e-11	7.4e-12	7.07e-11	5.72e-09

Table 6.4. Magnitude from the U-test for IoTvar

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
CPU	large	large	large	large	large	large	large	large
Energy	large	large	large	large	large	large	large	large

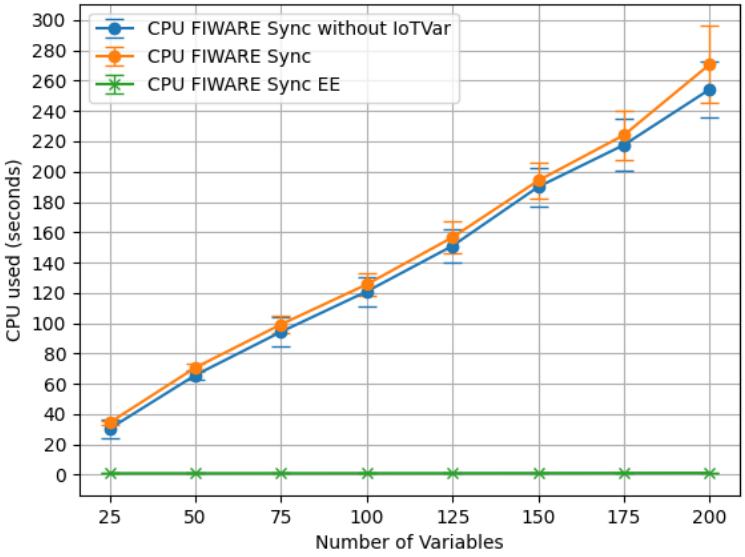


Figure 6.5. IoTvar CPU usage using EE strategies

6.4.3 IoTvar Energy Model calibration

IoTvar provides an internal energy model as shown in Section 6.3. Figure 6.6 shows the results of the tests with the Wattmeter along with an estimation of the energy consumption done by the model. The estimation follows a straight line that increases the more variables are declared in the code. It was calibrated through several tests until it was able to estimate the energy consumption without straying far from the values given by the tests done with a wattmeter. Table 6.5 shows the values used to be able to calculate the energy consumption. Furthermore, Table 6.6 shows the difference between the estimation given by the energy model and the energy consumption given by the wattmeter.

Variable	Value
C_V	0.02
C_{net}	90.0
M_{netS}	1.0
M_{netI}	10.0
C_{cpu}	87.0

Table 6.5. Values of constants and modifiers of the energy model.

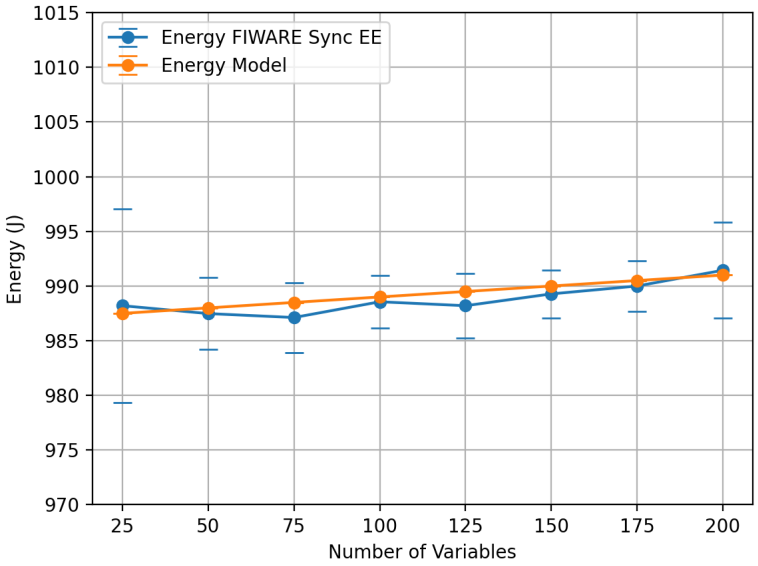


Figure 6.6. IoTvar energy consumption using energy model

Table 6.6. Estimation using the energy model and the wattmeter

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
Energy Wattmeter	988.2 (±8.86)	987.48 (±3.3)	987.12 (±3.19)	988.56 (±2.43)	988.2 (±2.95)	989.28 (±2.2)	990.0 (±2.32)	991.44 (±4.39)
Energy model	987.5	988	988.5	989	989.5	990	990.5	991

6.4.4 Results with JoularJX

IoTvar so far has been tested using a digital wattmeter. While this method of testing can give good measures and the wattmeter can reliably give energy consumption data, other tools can be used. One of these tools is JoularJX, a java agent which can be attached to java applications and measure energy consumption (of the CPU and DRAM only) at the method level. Using this tool we can gather energy usage from only the methods that IoTvar uses and then get the total energy of the IoT application using IoTvar. With this experimentation, we wanted to analyze if the communication call stack was consuming CPU and DRAM significantly.

Figure 6.7 shows the data from both the wattmeter and JourlarJX. When comparing both, there is a clear gap in the energy consumption reported. While the energy consumption measured with the wattmeter, which is from the whole

computer minus the idle consumption is around 1000 Joules, the consumption of IoTvar reported by JoularJX is around half of it. Table 6.7 also reports the percentage change between using the wattmeter and JoularJX. It is important to highlight that the energy data gathered from JourlarJX is given by the RAPL interface from the CPU and DRAM, and thus the energy consumption generated by IoTvar when using other resources such as network card and the disk is not taken into account. Moreover, the values gathered from JoularJX were not precise and had a high standard deviation which can cause problems when trying to reach conclusions about the energy consumption in IoT middleware.

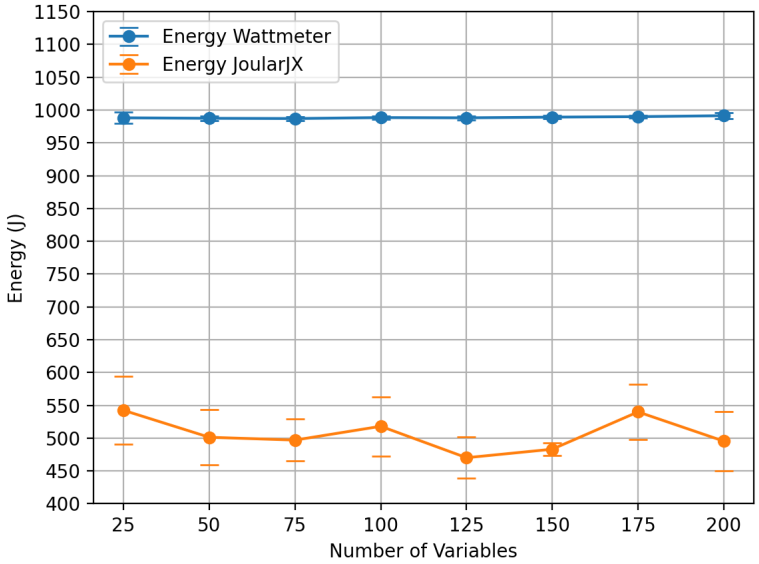


Figure 6.7. IoTvar energy consumption using JoularJX

Table 6.7. Difference between using the wattmeter and JoularJX

	Number of IoTvar variables							
Metric	25	50	75	100	125	150	175	200
Energy	45.11%	49.23%	49.65%	47.6%	52.43%	51.17%	45.46%	50.03%

In the first place, we thought that this gap was comparable to the results of [Econ06] that states that about 30-40% of the power is spent on the disk, the network, the I/O and peripherals, the power supplies, the regulators, and

the rest of the glue circuitry in the server that we can see with the Wattmeter but not with JoularJX. However, when studying the energy consumption by method result, we found that the *getStackTrace()* method called by JoularJX every 10ms to analyze the consumption by method was consuming around 80% of the energy by itself. Some more investigation would have been necessary for giving conclusions from this experimentation.

6.5 Discussion

The results of the tests have shown that the usage of energy-efficient strategies is lowering significantly the amount of energy consumed by IoTvar, especially when the number of variables enables a good level of grouping. Also, the energy awareness implemented enables IoTvar to have further knowledge of the energy context of the application (network status, variables energy consumption, etc). This energy context information is then used to provide enough information to build an energy model which is used to estimate the energy consumption of the middleware.

Nonetheless, there are still some limitations to the results: (i) the tests were run using only the WiFi interface, leaving a gap to other types of communication such as 3G and 4G which are popular in IoT consumer applications; (ii) the models and tests done are specific to the computer explained in Section 6.4.1, thus reproducing the results could be a hard task; and (iii) the tests were made to be comparable with the tests of the IoTvar middleware without the energy-efficient/aware strategies to provide comparable results, which limits the statistical analysis to a smaller number of variables than the IoTvar grouping strategy can support.

IoTvar supports multiple platforms, each with its own set of functionalities and specific APIs. The grouping mechanism provided by IoTvar was possible due to the implementation of this capability in the FIWARE platform. This availability is not yet available in other platforms, integrating this requirement in other platforms could be the first step toward energy efficiency in IoT consumer applications.

6.6 Conclusion

The energy consumption of IoT middleware is an important feature when IoT applications consider using them. By using energy-efficient and aware strategies in IoT middleware, applications with multiple variables can improve their

energy efficiency. Moreover, the implementation of awareness over the consumption of the interactions through an energy model resulted in the provision of an energy budget feature by IoTvar for IoT applications to lower their energy consumption. The energy model may also be used in the future to bring energy awareness to end-users as well as to the designers of applications.

This chapter has presented the energy-efficient/aware strategies, the architecture of the IoTvar middleware, and how they were implemented to lower energy consumption. The strategies were evaluated in an application using the middleware and the results have shown that it does lower energy consumption. Concerning the statistical analysis, there is a clear difference between using and not using these strategies with the values reaching a maximum percentage change of around 60% less energy consumption when using energy-efficient strategies. Finally, other energy consumption strategies could also have been included: interaction pattern switch, modifying the freshness for grouping more variables, and switch of communication protocols. They would need to be evaluated to confirm their efficiency, but could further increase the energy efficiency in IoT middleware.

Conclusions and future works

Chapter 7

Conclusions and future work

This chapter presents the conclusions of this work. The contributions with a summary of this thesis and the final remarks for the propositions of energy-efficiency/awareness in IoT middleware are presented in Section 7.1. Finally, future directions regarding energy-efficiency/aware IoT middleware are discussed in Section 7.2.

7.1 Conclusions

The world has been and is still experiencing a greater need for energy production caused by the evolution of technology and the expansion of many technological areas such as the digital. Energy usage in IoT in particular presents a significant challenge as the number of devices is increasing over the years. As a consequence, many IoT applications are being developed and they must interact with the heterogeneous components of an IoT system. Those IoT applications need to be energy-efficient/aware to benefit the reduction of energy consumption and also to be able to run on energy-constrained devices.

The complexity of IoT applications in running energy-efficient/aware strategies is also a concern as they need to deal with heterogeneity and a huge amount of data. Aiming to promote reducing the complexity of introducing energy efficiency/awareness into IoT applications, this thesis proposes an IoT middleware to help deal with communication abstractions, heterogeneity, and the overall integration among components of an IoT system.

The first contribution of this thesis is an IoT middleware for IoT consumer applications. IoTvar is proposed as this IoT middleware provides application developers with a way of interacting with IoT platforms at low development cost. For this purpose, IoTvar encompasses proxies representing IoT platform

virtual entities. These proxies handle the complexity of interacting with the IoT platform in both synchronous and publish-subscribe ways. Additionally, IoTvar offers a way to bypass the need of understanding the IoT platform-specific API and data model. For this contribution, we have described not only the integration of IoTvar with the FIWARE, OM2M, and muDEBS IoT platforms but also showed how the architecture of the middleware is composed and how it can be expanded for other platforms.

The evaluation of IoTvar was performed with all three supported platforms addressing the balance between the relative cost of IoTvar for both synchronous and publish-subscribe handling of information as well as the benefits for developers. Concerning the cost, the results of the statistical analysis revealed that there is an impact, in most cases, on CPU, memory, and energy consumption. Furthermore, the results show a global difference of around 5% of increase when using IoTvar. However, these values, although higher, could be acceptable depending on the application that uses the middleware.

The second contribution is the analysis of the energy consumption of the HTTP and MQTT protocols for IoT consumer applications. We have measured the energy consumption on user devices connected with WiFi (802.11n). The results show that, for the same amount of received observations, the publish/subscribe interaction pattern has lower energy consumption (around 92% lower) than the synchronous interaction pattern. Concerning the publish/-subscribe interaction pattern, MQTT consumes less than the HTTP protocol (around 20% less). Furthermore, the payload has a low impact on energy consumption having a 9% overhead from 24 to 3120 bytes payloads. The results give guidelines that could be used not only for IoT applications, which need to be more energy-efficient but also for energy-efficiency in IoT middleware.

The third contribution of this thesis is energy efficiency and energy awareness inside an IoT middleware dedicated to consumer applications. The work done has presented applicable strategies, and the architecture of IoTvar with support for energy efficiency/awareness, thereby introducing new components that enabled the strategies to lower the energy consumption to be developed and then used by an IoT application. These strategies came from message grouping, refresh-time adaptation, and energy budget management strategies implemented in the middleware.

The energy-efficient IoTvar middleware was statistically evaluated by comparing the usage of an IoT application with and without the strategies. The results show that: (i) the use of message grouping, message rate control, and energy models can decrease energy consumption by close to 60% and, (ii) the energy model can be calibrated for each device to show an estimation similar

to the true energy consumption given by a wattmeter. The energy model may provide energy awareness to end users and application designers. Furthermore, we also have proposed the usage of an energy budget mechanism to further enhance the energy efficiency in IoTvar by providing another way of controlling the grouping and refresh rate of the variables.

7.2 Future Work

The importance of energy efficiency/awareness in IoT middleware raises some points for future work. Below we highlight some perspectives of this work and discuss them.

- The cost of communication in middleware is still a subject to be studied. When evaluating the energy consumption of software, different interfaces need to be measured to have a more specific consumption regarding the data exchanged between the middleware and other IoT system components. Moreover, even when measuring, for example, a network interface, the brand of the hardware used impacts and the cost of the communication can vary. Thus, future research is needed to measure the cost of communication over different interfaces used in the device and separate the energy consumption of each of them.

- The energy-efficient/aware strategies developed for IoT middleware was conceived to be used in a domain where there is a high volume of communication with distributed IoT objects. These strategies could also be applied to any domain where energy consumption is an issue and there is a need to lower the energy used when communicating with different software in a network. Future research on the topic is to employ those strategies in other domains where the energy strategies along with middleware can be applied such as big data and artificial intelligence.

- The evaluation was done for IoTvar with the energy-efficient/aware strategies considering an IoT consumer application working in a single client. The future of this solution is to support distributed applications and enable IoT middleware-level cooperation to provide energy awareness for a multi-component system. This could be done by implementing communication over the network between the different IoT applications through the IoT middleware running inside the application or an external middleware that will coordinate the communication

and exchange of energy information. This information is related to network conditions, CPU usage, energy budget, energy models, and other resources that would be relevant to the energy efficiency/awareness of the IoT middleware. Furthermore, it is imperative to evaluate and show the impacts of such energy-efficient/aware features.

- The awareness provided by the IoTvar middleware is automatically done with no input from the user and only programmable, at first, by the developer of the IoT application. Shah et. al. explains that "the maintenance of the balance between the comfort index and power consumption is also a significant issue" [Shah19], and as awareness influences the energy-efficiency, which then has an impact on the quality of the applications (e.g. freshness), then it can not be transparent, or at least keep the users in the loop and knowing what is happening in the application. The energy awareness of the user may, in turn, have a positive impact on energy consumption in the future.

Bibliography

[Aaza20] Mohammad Aazam, Saif Ul Islam, Salman Tariq Lone, and Assad Abbas. Cloud of Things (CoT): Cloud-Fog-IoT Task Offloading for Sustainable Internet of Things. *IEEE Transactions on Sustainable Computing*, pages 1–1, 2020.

[Aaza21] Mohammad Aazam, Sherali Zeadally, and Eduardo Feo Flushing. Task Offloading In Edge Computing for Machine Learning-based Smart Healthcare. *Computer Networks*, 191:108019, 2021.

[Akke16] S. Akkermans, R. Bachiller, N. Matthys, W. Joosen, D. Hughes, and M. Vučinić. Towards Efficient Publish-subscribe Middleware in the IoT with IPv6 multicast. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016.

[AM18] Basem M. Al-Madani and Essa Q. Shahra. An Energy Aware Plateform for IoT Indoor Tracking Based on RTPS. *Procedia Computer Science*, 130:188–195, 2018. The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops.

[Amar16] Leonardo Amaral, Everton de Matos, Ramão Tiburski, F. Hessel, Willian T. Lunardi, and Sabrina Marczak. *Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G*, pages 333–367. 04 2016.

[Andr15] Anders S. G. Andrae and Tomas Edler. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges*, 6(1):117–157, 2015.

[AR19] Anas Al-Roubaiey, Tarek Sheltami, Ashraf Mahmoud, and Ansar Yasar. EATDDS: Energy-aware middleware for wireless sensor and

actuator networks. *Future Generation Computer Systems*, 96:196–206, 2019.

[Asgh18] Parvaneh Asghari, Amir Rahmani, and Hamid Haj Seyyed Javadi. Internet of Things applications: A Systematic Review. *Computer Networks*, 148:241–261, 12 2018.

[Atti19] Tarek M. Attia. Challenges and Opportunities in the Future Applications of IoT Technology. In *2nd Europe - Middle East - North African Regional Conference of the International Telecommunications Society (ITS)*, Calgary, 2019. International Telecommunications Society (ITS).

[Atzo10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[Band13] Soma Bandyopadhyay and Abhijan Bhattacharyya. Lightweight Internet Protocols for Web Enablement of Sensors Using Constrained Gateway Devices. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 334–340, 2013.

[Bano17] Y. Banouar, T. Monteil, and C. Chassot. Analytical Model for adaptive QoS Management at the Middleware level in IoT. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 1201–1208, 2017.

[Beng12] Pierre-Jean Benghozi, Sylvain Bureau, and Françoise Massit-Folléa. *Internet of Things: What Challenges for Europe*. Les Editions de la MSH, 2012.

[Benh22] Sana Benhamaid, Abdelmadjid Bouabdallah, and Hicham Lakhlef. Recent Advances in Energy Management for Green-IoT: An up-to-date and comprehensive survey. *Journal of Network and Computer Applications*, 198:103257, 2022.

[Bens08] Djamal Benslimane, Schahram Dustdar, and Amit Sheth. Services Mashups: The New Generation of Web Applications. *Internet Computing, IEEE*, 12:13–15, 10 2008.

[Bere07] Gian Paolo Beretta. World Energy Consumption and Resources: An Outlook for the Rest of the Century. *Int. J. Environmental Technology and Management*, 7, 01 2007.

[Bian12] A. P. Bianzino, C. Chaudet, D. Rossi, and J. Rougier. A Survey of Green Networking Research. *IEEE Communications Surveys Tutorials*, 14(1):3–20, 2012.

[Blai16] Gordon S. Blair, Douglas C. Schmidt, and Chantal Taconet. Middleware for Internet Distribution in the Context of Cloud Computing and the Internet of Things - Editorial Introduction. *Ann. des Télécommunications*, 71(3-4):87–92, 2016.

[Borg19] Pedro Victor Borges, Chantal Taconet, Sophie Chabridon, Denis Conan, Thais Batista, Everton Cavalcante, and Cesar Batista. Mastering Interactions with Internet of Things Platforms through the IoTVar Middleware. *Proceedings*, 31(1), 2019.

[Bouj14] Raja Boujbel, Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Jean-Paul Arcangeli, and Claire Lecocq. MuScADeL: A Deployment DSL Based on a Multiscale Characterization Framework. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops*, pages 708–715, 2014.

[Boul19] Georgios Bouloukakis, Nikolaos Georgantas, Patient Ntumba, and Valérie Issarny. Automated Synthesis of Mediators for Middleware-layer Protocol Interoperability in the IoT. *Future Gener. Comput. Syst.*, 101:1271–1294, 2019.

[Bour13] Aurélien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. PowerAPI: A Software Library to Monitor the Energy Consumed at the Process-Level. *ERCIM News*, 92:43–44, January 2013.

[Camp99] Andrew T Campbell, Geoff Coulson, and Michael E Kounavis. Managing Complexity: Middleware explained. *IT professional*, 1(5):22–28, 1999.

[Cane22] Rodrigo Canek, Pedro Borges, and Chantal Taconet. Analysis of the Impact of Interaction Patterns and IoT Protocols on Energy Consumption of IoT Consumer Applications. In *DAIS 2022: 17th International Conference on Distributed Applications and Interoperable Systems*, Lecture Notes in Computer Science, pages 1–17, Lucca, Italy, June 2022. Springer.

[Cecc19] Cyril Cecchinel, François Fouquet, Sébastien Mosser, and Philippe Collet. Leveraging Live Machine Learning and Deep Sleep to Support a Self-adaptive Efficient Configuration of Battery Powered Sensors. *Future Generation Computer Systems*, 92:225–240, 2019.

[Chao11] H. Chao, Y. Chen, and J. Wu. Power Aaving for Machine to Machine Communications in Cellular Networks. In *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pages 389–393, 2011.

[Chaq12] M. A. Chaqfeh and N. Mohamed. Challenges in Middleware Solutions for the Internet of Things. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 21–26, 2012.

[Cout05] Joëlle Coutaz, James L Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[dbHF19] Shift Project directed by Hugues Ferreboeuf. Lean ICT – Towards digital sobriety. https://theshiftproject.org/wp-content/uploads/2019/03/Lean-ICT-Report_The-Shift-Project_2019.pdf, 2019.

[Deli13] Flávia C. Delicato, Paulo F. Pires, and Thais Batista. *The Programming and Execution Module (PEM)*, pages 45–55. Springer London, London, 2013.

[Deni20] N. Denis, P. Chaffardon, D. Conan, M. Laurent, S. Chabridon, and J. Leneutre. Privacy-preserving Content-based Publish/Subscribe with Encrypted Matching and Data Splitting. In *Proc. of the 17th International Joint Conference on e-Business and Telecommunications*, pages 405–414, Paris, France, July 2020. INSTICC, SciTePress.

[Diaz08] Dustin Diaz and Ross Harmes. *The Proxy Pattern*, pages 197–214. Apress, Berkeley, CA, 2008.

[Dizd19] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. volume 51, New York, NY, USA, January 2019. Association for Computing Machinery.

[dO20] Egberto Armando de Oliveira, Flávia Delicato, and Marta Mattoso. An Energy-aware Data Cleaning Workflow for Real-time Stream Processing in the Internet of Things. In *Anais do IV Workshop de Computação Urbana*, pages 71–83, Porto Alegre, RS, Brasil, 2020. SBC.

[Dybå07] Tore Dybå, Torgeir Dingsøyr, and Geir K. Hanssen. Applying systematic reviews to diverse study types: An experience report. In *First International Symposium on Empirical Software Engineering and Measurement*, pages 225–234, 2007.

[EAI16] Energy Efficiency of the Internet of Things - Technology and Energy Assessment Report. https://www.iea-4e.org/wp-content/uploads/publications/2016/04/Energy_Efficiency_of_the_Internet_of_Things_-_Technical_Report_FINAL.pdf, April 2016.

[Econ06] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system Power Analysis and Modeling for Server Environments. *International Symposium on Computer Architecture (IEEE)*, 01 2006.

[Elha21] Abdessalam Elhabbash and Yehia Elkhatib. Energy-Aware Placement of Device-to-Device Mediation Services in IoT Systems. In *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings*, page 335–350, Berlin, Heidelberg, 2021. Springer-Verlag.

[Este15] C. Estevez and J. Wu. Recent Advances in Green Internet of Things. In *2015 7th IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–5, 2015.

[FIWA] FIWARE-NGSI v2 Specification. http://telefonicaid.github.io/fiware-orion/api/v2/stable/.

[Geor07] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically Rigorous Java Performance Evaluation. *SIGPLAN Not.*, 42(10):57–76, oct 2007.

[Hass09] Mohamed G. Hassan, R. Hirst, C. Siemieniuch, and A.F. Zobaa. The Impact of Energy Awareness on Energy Efficiency. *International Journal of Sustainable Engineering*, 2(4):284–297, 2009.

[Heja18] Hamdan Hejazi, Husam Rajab, Tibor Cinkler, and László Lengyel. Survey of Platforms for Massive IoT. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pages 1–8. IEEE, 2018.

[Hess04] Melinda Hess and Jeffrey Kromrey. Robust Confidence Intervals for Effect Sizes: A Comparative Study of Cohen's d and Cliff's Delta Under Non-normality and Heterogeneous Variances. *Paper Presented at the Annual Meeting of the American Educational Research Association*, 01 2004.

[Hofe18] Johannes Hofer and Sachin Pawaskar. Impact of the Application Layer Protocol on Energy Consumption, 4G Utilization and Performance. In *2018 3rd Cloudification of the Internet of Things (CIoT)*, pages 1–7, 2018.

[Huan14] Zhenqiu Huang, Kwei-Jay Lin, and Lina Han. An energy sentient methodology for sensor mapping and selection in IoT systems. In *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pages 1436–1441, 2014.

[Infs08] D Infso. Networked Enterprise & RFID INFSO G. 2 Micro & Nanosystems, in co-operation with the Working Group RFID of the ETP EPOSS, Internet of Things in 2020, Roadmap for the Future [R]. *Information Society and Media, Tech. Rep*, 10, 2008.

[Jala12] S. Jalali and C. Wohlin. Systematic literature studies: Database searches vs. backward snowballing. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 29–38, 2012.

[Jeon17] Soobin Jeon and Inbum Jung. MinT: Middleware for Cooperative Interaction of Things. *Sensors*, 17(6), 2017.

[Josh17] Jetendra Joshi, Vishal Rajapriya, S.R. Rahul, Pranith Kumar, Siddhanth Polepally, Rohit Samineni, and D.G. Kamal Tej. Performance Enhancement and IoT Based Monitoring for Smart Home. In *2017 International Conference on Information Networking (ICOIN)*, pages 468–473, 2017.

[Kalb18] Tomasz Kalbarczyk and Christine Julien. Omni: An Application Framework for Seamless Device-to-Device Interaction in the Wild. In

Proceedings of the 19th International Middleware Conference, Middleware '18, page 161–173, New York, NY, USA, 2018. Association for Computing Machinery.

[Kicz01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An Overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, pages 327–353, Berlin, Heidelberg, 2001. Springer-Verlag.

[Kitc11] Barbara A. Kitchenham, David Budgen, and O. [Pearl Brereton]. Using mapping studies as the basis for further research – A participant-observer case study. *Information and Software Technology*, 53(6):638 – 651, 2011. Special Section: Best papers from the APSEC.

[Kitc16] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-Based Software Engineering and systematic reviews*. Chapman and Hall/CRC Press, USA, 2016.

[Lann13] Bart Lannoo, Sofie Lambert, WV Heddeghem, Mario Pickavet, Fernando Kuipers, George Koutitas, Harris Niavis, Anna Satsiou, Michael Till Beck, Andreas Fischer, et al. Overview of ICT Energy Consumption (deliverable 8.1). *EU Project FP7-2888021, European Network of Excellence in Internet Science*, 2013.

[Lee15] In Lee and Kyoochun Lee. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431 – 440, 2015.

[Li14] Wei Li, Flávia C. Delicato, Paulo F. Pires, Young Choon Lee, Albert Y. Zomaya, Claudio Miceli, and Luci Pirmez. Efficient Allocation of Resources in Multiple Heterogeneous Wireless Sensor Networks. *Journal of Parallel and Distributed Computing*, 74(1):1775–1788, 2014.

[Li15] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: A Survey. *Information Systems Frontiers*, 17(2):243–259, April 2015.

[Mahm04] Qusay H Mahmoud. *Middleware for communications*, volume 73. Wiley Online Library, 2004.

[Mann47] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60, 1947.

[Mari14] Pierrick Marie, Léon Lim, Atif Manzoor, Sophie Chabridon, Denis Conan, and Thierry Desprats. QoC-Aware Context Data Distribution in the Internet of Things. M4IOT '14, pages 13–18, New York, NY, USA, 2014. Association for Computing Machinery.

[Marq17] Gonçalo Marques, Nuno Garcia, and Nuno Pombo. *A Survey on IoT: Architectures, Elements, Applications, QoS, Platforms and Security Concepts*, pages 115–130. Springer International Publishing, Cham, 2017.

[Mine16] Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. A Gap Analysis of Internet-of-Things Platforms. *Computer Communications*, 89-90:5–16, September 2016.

[Mogh15] Fahimeh Alizadeh Moghaddam, Patricia Lago, and Paola Grosso. Energy-Efficient Networking Solutions in Cloud-Based Environments: A Systematic Literature Review. *ACM Comput. Surv.*, 47(4), May 2015.

[Mukh20] Amartya Mukherjee, Nilanjan Dey, and Debashis De. EdgeDrone: QoS aware MQTT middleware for mobile edge computing in opportunistic Internet of Drone Things. *Computer Communications*, 152:93 – 108, 2020.

[Muno19] Daniel-Jesus Munoz, José A. Montenegro, Mónica Pinto, and Lidia Fuentes. Energy-aware environments for the development of green applications for cyber–physical systems. *Future Generation Computer Systems*, 91:536 – 554, 2019.

[Nakh15] Bhumi Nakhuva and Tushar Champaneria. Study of Various Internet of Things Platforms. *International Journal of Computer Science & Engineering Survey*, 6(6):61–74, 2015.

[NGSI] FIWARE-NGSI v2 Specification. https://fiware.github.io/specifications/ngsiv2/stable/.

[Ngu17] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017.

[Niel99] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[Niko20] Neven Nikolov. Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems. In *2020 XXIX International Scientific Conference Electronics (ET)*, pages 1–4, 2020.

[NODE21] Node-Red. https://nodered.org/, 2021. Accessed on 21-05-2021.

[Nour12] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. A Preliminary Study of the Impact of Software Engineering on GreenIT. In *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pages 21–27, 2012.

[Nour13] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A Review of Energy Measurement Approaches. *SIGOPS Oper. Syst. Rev.*, 47(3):42–49, nov 2013.

[Nour22] Adel Noureddine. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. In *18th International Conference on Intelligent Environments (IE2022)*, Biarritz, France, Jun 2022.

[OASI15] OASIS. MQTT Version 3.1.1 Plus Errata 01. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf, 12 2015. Accessed on 21-05-2021.

[Okra15] Tom Okrasinski, Tim Fleming, Amanda Moore, Gabrielle Gines, Steve Kelly, Steven Moore, and Mark Shackleton. Smarter 2030. Technical report, Global eSustainability Initiative (GeSI), Brussels, Belgium, 2015.

[OM2M] the oneM2M REST APIs. https://www.onem2m.org/getting-started/onem2m-overview/application-program-interfaces-api.

[oneM] oneM2M. Who we are. https://www.onem2m.org/harmonization-m2m.

[Onor18] Uviase Onoriode and Gerald Kotonya. IoT Architectural Framework: Connection and Integration Framework for IoT Systems. *Electronic Proceedings in Theoretical Computer Science*, 264:1–17, 02 2018.

[Padh17] Satyajit Padhy, Hsin-Yu Chang, Ting-Fang Hou, Jerry Chou, Chung-Ta King, and Cheng-Hsin Hsu. A Middleware Solution for Optimal Sensor Management of IoT Applications on LTE Devices. In Jong-Hyouk Lee and Sangheon Pack, editors, *Quality, Reliability, Security and Robustness in Heterogeneous Networks*, volume 199 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 283–292. Springer International Publishing, Switzerland, 2017.

[Pang16] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What Do Programmers Know about Software Energy Consumption? *IEEE Software*, 33(03):83–89, may 2016.

[Pasr18] S. Pasricha. Overcoming Energy and Reliability Challenges for IoT and Mobile Devices with Data Analytics. In *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, pages 238–243, 2018.

[Pate15] Pankesh Patel and Damien Cassou. Enabling High-level Application Development for the Internet of Things. *Journal of Systems and Software*, 103:62–84, 2015.

[PC03] G. Pardo-Castellote. OMG Data-Distribution Service: Architectural Overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206, 2003.

[Pete08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, page 68–77, Swindon, GBR, 2008. BCS Learning and Development Ltd.

[PZ13] Ivana Podnar Zarko, Aleksandar Antonic, and Krešimir Pripužic. Publish/Subscribe Middleware for Energy-Efficient Mobile Crowdsensing. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, page 1099–1110, New York, NY, USA, 2013. Association for Computing Machinery.

[Rama16] Gowri Ramachandran, José Proença, Wilfried Daniels, Mario Pickavet, Dimitri St, Dimitri Staessens, Christophe Huygens, Wouter

Joosen, and Danny Hughes. Hitch Hiker 2.0: a binding model with flexible data aggregation for the Internet-of-Things. *Journal of Internet Services and Applications*, 7, 04 2016.

[Ray16] Partha Pratim Ray. A Survey of IoT Cloud Platforms. *Future Computing and Informatics Journal*, 1(1):35 – 46, 2016.

[Razz16] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[Reev13] April Reeve. Chapter 12 - Data Integration Patterns. In April Reeve, editor, *Managing Data in Motion*, MK Series on Business Intelligence, pages 79–85. Morgan Kaufmann, Boston, 2013.

[Rote12] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro*, 32(2):20–27, 2012.

[Sark16] Chayan Sarkar, Vijay S. Rao, R. Venkatesha Prasad, Sankar Narayan Das, Sudip Misra, and Athanasios Vasilakos. VSF: An Energy-Efficient Sensing Framework Using Virtual Sensors. *IEEE Sensors Journal*, 16(12):5046–5059, 2016.

[Shah19] Abdul Shah, Haidawati Nasir, Muhammad Fayaz, Adidah Lajis, and Asadullah Shah. A Review on Energy Consumption Optimization Techniques in IoT Based Smart Building Environments. *Information*, 10(3):108, Mar 2019.

[Shai17a] F. K. Shaikh, S. Zeadally, and E. Exposito. Enabling Technologies for Green Internet of Things. *IEEE Systems Journal*, 11(2):983–994, 2017.

[Shai17b] Faisal Karim Shaikh, Sherali Zeadally, and Ernesto Exposito. Enabling Technologies for Green Internet of Things. *IEEE Systems Journal*, 11(2):983–994, 2017.

[Shap65] Samuel Sanford Shapiro and Martin Wilk. An Analysis of Variance Test for Normality. *Biometrika*, 52(3-4):591–611, 1965.

[Shap86] Marc Shapiro. Structure and Encapsulation in Distributed Systems: The Proxy Principle. In *Int. Conf. on Distr. Comp. Sys. (ICDCS)*, Int. Conf. on Distr. Comp. Sys. (ICDCS), pages 198–204, Cambridge, MA, USA, United States, 1986. IEEE.

[Shek19] Shashank Shekhar, Ajay Chhokra, Hongyang Sun, Aniruddha Gokhale, Abhishek Dubey, and Xenofon Koutsoukos. URMILA: A Performance and Mobility-Aware Fog/Edge Resource Management Middleware. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 118–125, 2019.

[Shen15] Z. Sheng, C. Mahapatra, C. Zhu, and V. C. M. Leung. Recent Advances in Industrial Wireless Sensor Networks Toward Efficient Management in IoT. *IEEE Access*, 3:622–637, 2015.

[Sing17] K. J. Singh and D. S. Kapoor. Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine*, 6(2):57–68, 2017.

[Song17] Zheng Song, Minh Le, Young-Woo Kwon, and Eli Tilevich. Extemporaneous Micro-Mobile Service Execution Without Code Sharing. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 181–186, 2017.

[Soun08] Karthik Soundararajan and Robert Brennan. Design Patterns for Real-time Distributed Control System Benchmarking. *Robotics and Computer-integrated Manufacturing - ROBOT COMPUT-INTEGR MANUF*, 24:606–615, 10 2008.

[Sutr17] Pierre Sutra, Etienne Rivière, Cristian Cotes, Marc Sánchez Artigas, Pedro Garcia Lopez, Emmanuel Bernard, William Burns, and Galder Zamarreño. CRESON: Callable and Replicated Shared Objects over NoSQL. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 115–128, 2017.

[Told19] Jevgenijus Toldinas, Borisas Lozinskis, Edgaras Baranauskas, and Algirdas Dobrovolskis. MQTT Quality of Service versus Energy Consumption. In *2019 23rd International Conference Electronics*, pages 1–4, 2019.

[Trei10] Jan Treibig, Georg Hager, and Gerhard Wellein. Likwid: A lightweight Performance-oriented Tool Suite for x86 Multicore Environments. In *2010 39th international conference on parallel processing workshops*, pages 207–216. IEEE, 2010.

[Varg00] András Vargha and Harold D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[VH14] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. Trends in Worldwide ICT Electricity Consumption from 2007 to 2012. *Computer Communications*, 50:64–76, 2014.

[Vill10] Norha M Villegas and Hausi A Müller. Managing Dynamic Context to Optimize Smart Interactions and Services. *The smart internet*, pages 289–318, 2010.

[Wang16] K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu. Green Industrial Internet of Things Architecture: An Energy-Efficient Perspective. *IEEE Communications Magazine*, 54(12):48–54, 2016.

[Wei14] Z. Wei and D. Q. Ren. Review of Energy Aware Big Data Computing Measurements, Benchmark Methods and Performance Analysis. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–4, 2014.

[WIRE21] Wirecloud. https://wirecloud.readthedocs.io/en/stable/, 2021. Accessed on 21-05-2021.

[Wu16] J. Wu, S. Guo, J. Li, and D. Zeng. Big Data Meet Green Challenges: Big Data Toward Green Applications. *IEEE Systems Journal*, 10(3):888–900, 2016.

[Wu18] J. Wu, S. Guo, H. Huang, W. Liu, and Y. Xiang. Information and Communications Technologies for Sustainable Development Goals: State-of-the-Art, Needs and Perspectives. *IEEE Communications Surveys Tutorials*, 20(3):2389–2406, 2018.

[Yoct] YoctoPuce. Who are we? https://www.yoctopuce.com/EN/aboutus.php. Accessed on 17-10-2021.

Part III

Appendix

Summary of selected primary studies in the SLR

Study **S1** considers different task offloading strategies. Tasks are the processes that transfer data from connected IoT objects to a deployed fog, cloud, or their collaboration (FC). A gateway might also be used to communicate between fog, cloud, and the FC. The proposed strategies use either a random (asynchronous multithreaded) or a first-come/first-served (synchronous single-threaded) task distribution. The fog nodes can communicate with each other to better distribute the tasks. When using the gateway, the decision is to send large datasets to the cloud and small datasets to the fog. The study also concludes that using a gateway with the FC strategy provides better results in terms of energy consumption.

Study **S2** proposes a microservice-based middleware that distributes executable code across nearby mobile devices. The mobile device that deploys the code is selected based on the available device's resources. The study compares the service initiation time (the first use of the service), executing JavaScript or native services that provide image processing, Internet sharing, and GPS sharing for mobile devices across the network. Results from an empirical evaluation show that executing services on nearby devices can improve the energy consumption of the mobile device that calls for the execution of these services.

Study **S3** proposes Omni, a device-to-device middleware with the periodic adaptive discovery of neighbor devices using lightweight discovery mechanisms in wireless local area networks. Discovered devices are connected when data needs to be transferred, and the communication technology can change to the methods chosen by the application developer according to the volume of these data. The authors report an evaluation regarding energy consumption and latency on the discovery technology in different scenarios. They conclude that changing the protocol for different amounts of data and finding the best discovery technology can improve energy efficiency.

Study **S4** proposes EatDDS, an energy-aware data distribution service

(DDS) dedicated to wireless sensor networks (WSN). EatDDS aims to equitably share the consumption among the nodes to improve the TinyDDS energy-efficient protocol. A TinyOS simulator for WSN is modified to become energy-aware in the study. Simulation results show that using EatDDS can improve network lifetime compared to TinyDDS. The network lifetime can be seen as energy efficiency in battery-constrained environments since the goal is to reduce the energy utilization to keep the IoT system working as long as possible.

Study **S5** proposes an adaptation of a publish-subscribe middleware by adding a layer between the broker and the client applications (in an IPv6 network) to send notifications via IPv6 multicast rather than using several point-to-point messages. The proposed framework maps application-layer subscriber groups to network layer multicast groups. The proposal reduces (i) the network overhead when there are many consumers for the same group on the same network (edge side) and (ii) the energy consumption on the edge broker node.

Study **S6** proposes URMILA (*Ubiquitous Resource Management for Interference and Latency-Aware services*). This middleware makes effective tradeoffs between using fog and edge resources while ensuring that the latency requirements of the IoT applications are met. URMILA works by transferring tasks from applications to fogs when the client needs to reduce the processing on the end-user device and thus reduce the energy consumption. It determines if the request will be processed locally or remotely on the selected fog server. The study shows that using URMILA contributes to meeting application requirements while introducing energy efficiency.

Study **S7** proposes MinT (*Middleware for Cooperative Interaction of Things*), a middleware in which IoT devices directly connect to peripheral devices and construct a local or global network where they share data in an energy-efficient way. MinT provides an abstract layer, a system layer, an interaction layer, and a high-level API for applications. The study reports an evaluation comparing the proposed solution with an existing message processing middleware and shows that MinT can reduce IoT devices' latency and power consumption.

Study **S8** proposes determining an optimal configuration of sensors for extending their battery life. The solution is to optimize sensor usage, network usage, and measurement quality in terms of (i) configuring the sensors' sampling frequency with Machine Learning and (ii) optimizing network usage according to the frequency of requests from the deployed software applications. The study shows measurements according to the sensors' battery life and how

their solution improves the lifetime of a WSN.

Study **S9** proposes a middleware to minimize the total energy consumption of an IoT application while ensuring that the requested accuracy is met. The middleware intends to find the sensors that consume the minor energy while satisfying the sensing requirements and maximizing the overall accuracy under an energy budget. Simulations demonstrate how the algorithm works and indicate that the proposed algorithms outperform some existing solutions.

Study **S10** proposes a data stream processing workflow to be deployed at the network's edge to (i) provide an energy-aware data collection component to reduce the network traffic, (ii) implement a density-based clustering component to efficiently perform the data cleaning task by quickly identifying and removing outliers from the data stream, and (iii) deliver a curated secondary data stream output that can be consumed by business applications, services or additional workflow tasks with real-time processing requirements. The study addresses the reduction of device power consumption, the enforcement of data accuracy and completeness, and real-time responses.

Study **S11** proposes a data analytics middleware for energy-efficient execution of various applications on commodity mobile devices. The proposed solution is to apply different Machine Learning algorithms to show different use cases and how the chosen algorithm impacts the energy efficiency of the IoT system. The middleware's energy efficiency was experimented in different scenarios considering battery lifetime, accuracy, response time, and adaptation to user behavior.

Study **S12** proposes the *Virtual Sensing Framework* (VSF) to reduce the interactions among the nodes of a WSN and hence the network's energy consumption. The solution works by keeping some nodes in the network in a low-power sleep state and running a heuristic algorithm to select the best nodes to improve the lifetime of the WSN. The study uses three metrics to evaluate the performance, namely (i) the number of transmitted data packets, (ii) the accuracy of prediction, and (iii) the energy expenditure by the nodes.

Study **S13** proposes the usage of edge computing to provide smart and opportunistic healthcare (oHealth). The solution uses Machine learning-based task offloading in edge computing with real data traces for several healthcare and safety-related scenarios. An evaluation shows how energy-efficient this solution is compared to not using task offloading.

Study **S14** proposes a message transfer mechanism using drone nodes that participate as edge computing components. This transfer mechanism is introduced as enhanced MQTT and MQTT-SN. An evaluation of the solution with a real testbed showed that the enhancement of MQTT and MQTT-SN use less

energy among other resources. However, for the MQTT/MQTT-SN broker, there is an increase in memory consumption as a drawback, which could lead to problems when there are many nodes in the network.

Study **S15** proposes a middleware platform that manages heterogeneous WSNs and efficiently shares their resources to increase the network's lifetime. The solution is a new resource allocation algorithm called SACHSEN (*reSource AlloCation in Heterogeneous SEnsor Networks*) that control running application, distribute, and coordinate nodes in the execution of submitted sensing tasks in an energy-efficient and QoS-enabled way. They show in their evaluation that when compared to other algorithms, SACHSEN produces promising results in terms of both application performance and energy efficiency depending on the scenario of the WSN.

Study **S16** proposes a mobile crowdsensing application for community sensing where sensors and mobile devices work together to provide data of interest to observe and measure events across a large geographic area. The solution uses the MoPS middleware, which provides energy efficiency as it suppresses the transmission of sensor readings from Mobile Internet-connected Objects (MIOs) and filters out the data that are not needed by the application. The evaluation used real user traces and showed that the solution could potentially save energy as the number of data transferred from the cloud to the mobile applications and vice-versa is significantly lowered.

Study **S17** proposes an energy-aware platform for indoor tracking using the RSS (*Receive Signal Strength*) algorithm, which is used to measure values such as position, distance, and others to provide a multi-hop protocol used between publishers and subscribers. The solution uses the sensor Data Distribution Service (sDDS) middleware to interconnect all system components. The evaluation showed that RSS has good accuracy when calculating values such as distance and position among sensors, which could be acceptable depending on the application, and has low energy consumption.

Study **S18** proposes the autonomic management of QoS in an IoT middleware to optimize the energy consumption of IoT entities (gateway) while maintaining the QoS constraints of critical IoT applications. The solution uses the MAPE-K autonomic loop as an analytical model for the OM2M middleware platform entities to estimate their performance metrics (response time, queue size, etc.) depending on requests arrival rate and monitor components to detec QoS degradation. The evaluation showed that autonomic management could react to constraints of applications. Moreover, for cases requiring lower energy consumption, the solution is able to react and use energy-efficient mechanisms such as CPU core deactivation to lower the energy usage.

Study **S19** proposes using the WuKong middleware with a new sensor selection methodology to minimize the total communication energy consumption and balance the energy costs on devices over a network to increase the system lifetime. The solution will use a service collocation and sensor selection algorithm to minimize the energy consumption and balance the energy usage over a network, increasing its lifetime. The evaluation relied on simulation with a collocation algorithm, a sensor selection algorithm, or both simultaneously. The results have shown that, depending on the number of sensors, the energy saving can be high or low, but it reduces energy consumption to a certain degree and can increase the system's lifetime, even if for a small margin.

Study **S20** proposes to design and deploy data analytics and Machine Learning to improve the energy efficiency of IoT systems through middleware. The solution uses energy-efficient/aware mechanisms such as mobile-to-cloud offloading, user-interaction aware optimizations, and spatiotemporal context-aware optimization. The mobile-cloud offloading is used to support high-end mobile data processing applications and enable them to offload mobile computations to the cloud. The user-interaction aware optimization is provided by the AURA middleware, which takes advantage of user idle time between interaction events of the foreground application to optimize CPU and backlight energy consumption. The spatiotemporal context-aware optimization is used to transparently capture contextual data attributes (day, hour, etc.), spatial environment data (e.g., ambient light, Wi-Fi RSSI, 3G/4G signal strength), and device state (e.g., battery status, CPU utilization) in order to remove unnecessary data. The solution was evaluated by comparing many Machine Learning algorithms, showing the gains in energy saving of the used mechanisms.

Study **S21** proposes a component binding model called Hich Hiker that allows for data aggregation over communications in a network. The solution allows application developers to specify high-priority remote bindings that generate radio transmissions or low-priority remote bindings that communicate exclusively using the data aggregation overlay, resulting in no additional transmissions. The evaluation compared the solution model with other models with similar behavior and showed that using Hitch Hiker to route low-priority traffic reduces energy consumption.

Study **S22** proposes an algorithm to use the network structure to compute the placement of mediation services to minimize the energy consumed by the interactions between IoT devices by formulating this placement as an integer linear programming (ILP). The solution allows the energy consumption among many sensors of a network to be calculated and then to place mediation services based on environmental changes. The evaluation of this algorithm is

compared to other three algorithms and has shown that it minimizes the energy consumption.

ECOLE
DOCTORALE

Titre : Support des middleware pour la prise en compte de la consommation énergétique l'Internet des objets

Mots clés : Internet des objets, Informatique verte, Middleware, Systèmes distribués

Résumé : L'Internet des objets (IoT) se caractérise par une myriade de dispositifs et de composants logiciels géographiquement dispersés ainsi que par une grande hétérogénéité en termes de matériel, de format de données et de protocoles. Au cours des dernières années, les plateformes IoT ont été proposées pour fournir une variété de services aux applications, tels que la découverte de dispositifs, la gestion du contexte et l'analyse des données. Cependant, le manque de standardisation fait que chaque plateforme IoT propose ses propres abstractions, API et patrons d'interactions. Par conséquent, la programmation des interactions entre une application IoT consommatrice de données et une plateforme IoT est complexe, sujette à des erreurs et demande un niveau de connaissance de la plateforme IoT approfondi de la part des développeurs. Les intergiciels IoT peuvent atténuer cette hétérogénéité, ils doivent fournir des services pertinents et ainsi faciliter le développement des applications.

L'efficacité énergétique de la technologie numérique devenant une priorité, l'augmentation du nombre de systèmes IoT pose des problèmes énergétiques. Dans ce contexte, il est essentiel de concevoir soigneusement les interactions entre les applications IoT grand public et les plateformes IoT en tenant compte de l'efficacité énergétique. Les intergiciels IoT ne doivent pas uniquement considérer l'efficacité énergétique comme une exigence non fonctionnelle laissée à l'application, Au contraire, parce qu'ils sont utilisés par de nombreuses applications, l'efficacité énergétique doit être au cœur de leur conception.

Cette thèse présente trois contributions concernant l'efficacité énergétique et la sensibilisation à l'énergie dans les intergiciels IoT pour les applications IoT consommatrices de données. La première contribution est la proposition d'un intergiciel IoT appelé IoTvar qui abstrait les capteurs virtuels IoT dans des variables IoT qui sont automatiquement mises à jour par l'intergiciel. La deuxième contribution est l'évaluation de la consommation d'énergie des interactions entre les applications IoT grand public et les plateformes IoT via les protocoles HTTP et MQTT. Cette évaluation a conduit à la proposition de lignes directrices pour améliorer l'efficacité énergétique des interactions. La troisième contribution est la proposition de stratégies d'efficacité énergétique pour des middleware IoT. Ces stratégies ont été intégrées dans l'intergiciel IoTvar pour assurer l'efficacité énergétique, mais aussi la sensibilisation à l'énergie par le biais d'un modèle énergétique et la gestion d'un budget énergétique fonction des exigences des utilisateurs. Les implémentations de l'architecture middleware IoT, avec et sans stratégie d'efficacité énergétique, ont été évaluées, et les résultats montrent que nous avons une diminution allant jusqu'à 60

Title : Middleware support for energy awareness in the Internet of Things (IoT)

Keywords : Internet of Things, Green Computing, Middleware, Distributed Systems

Abstract : The Internet of Things (IoT) is characterized by a myriad of geographically dispersed devices and software components as well as high heterogeneity in terms of hardware, data, and protocols. Over the last few years, IoT platforms have been used to provide a variety of services to applications such as device discovery, context management, and data analysis. However, the lack of standardization makes each IoT platform come with its abstractions, APIs, and interactions. As a consequence, programming the interactions between a consuming IoT application and an IoT platform is often time-consuming, error-prone, and depends on the developers' level of knowledge about the IoT platform. IoT middleware are proposed to alleviate such heterogeneity, provide relevant services, and ease application development.

As the energy efficiency of digital technology becomes a priority, the increase in IoT systems brings energy concerns. In this context, carefully designing interactions between IoT consumer applications and IoT systems with an energy-efficiency concern becomes essential. IoT middleware should not solely consider energy efficiency as a non-functional requirement. Instead, it needs to be at the solution's core as the middleware is expected to be shared by many applications and offer facilities to ease application development.

This work presents three contributions regarding energy-efficiency/awareness in IoT middleware for IoT consumer applications. The first contribution is the proposal of an IoT middleware for IoT consumer applications called IoTvar that abstracts IoT virtual sensors in IoT variables that are automatically updated by the middleware. The second contribution is the evaluation of the energy consumption of the interactions between IoT consumer applications and IoT platforms through the HTTP and MQTT protocols. This evaluation has led to the proposal of guidelines to improve energy efficiency when developing applications. The third contribution is the proposal of strategies for energy efficiency to be integrated into IoT middleware. Those strategies have been integrated into the IoTvar middleware to provide energy efficiency, but also energy awareness through an energy model and the management of an energy budget driven by user requirements. The implementations of the IoT middleware architecture, with and without energy-efficiency strategies, have been evaluated, and the results show that we have a difference of up to 60% the energy used by IoT applications by applying strategies to reduce energy consumption at the middleware level.

Institut Polytechnique de Paris
91120 Palaiseau, France