A Scalable and Secure System Architecture for Smart Buildings

THÈSE N^{O} 7905 (2017)

PRÉSENTÉE LE 20 OCTOBRE 2017
À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
GROUPE KAYAL
PROGRAMME DOCTORAL EN MICROSYSTÈMES ET MICROÉLECTRONIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Georgios LILIS

acceptée sur proposition du jury:

Dr A. Schmid, président du jury
Prof. M. Kayal, directeur de thèse
Dr A. Sanfilippo, rapporteur
Dr F. Lo Conte, rapporteur
Dr S.-R. Cherkaoui, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2017

What does not kill me, makes me stronger.
— Friedrich Nietzsche

Acknowledgements

The execution of any major project is a considerable undertaking which necessitates collaboration and support. This research work is certainly not an exemption. There are remarkable people behind this work to whom I am grateful, and that deserve to be acknowledged. It was a pleasure to work with, discuss, and receive their support during the past four years.

First of all, I would like to thank my thesis supervisor Prof. Maher Kayal for the excellent opportunity to conduct my research under his guidance and support. His trust and the research freedom throughout this process have been imperative for the success of this multidisciplinary engineering thesis. Moreover, I would like to acknowledge Dr. Fabrizio Lo Conte and Dr. Laurent Fabre from eSMART Technologies SA for their practical ideas, insights, and expertise that greatly assisted my research.

As a member of electronics laboratory, I greatly enjoyed the collaboration and brainstorming with my fellow colleagues. Firstly, I would like to thank my long-term officemates for the many moments we shared together while working closely on the Smart Building project, Gilbert Conus and Nastaran Asadi Zanjani. Secondly, I am especially grateful to Dr. Theodoros Kyriakidis and Dr. Guillaume Lanz with whom I have had the pleasure to work with; their advice, support, and the open-ended discussions we shared together are deeply appreciated. I would also like to mention and thank Olivier Van Cutsem, his research contributions, motivation, and feedback have been invaluable for the success of my research work. I would like finally to thank my good friends and former officemates who helped me get a great start in the postgraduate life: Dr. Lucian Barbut, Dr. Maria-Anna Chalkiadaki, Dr. Farzan Jazaeri, and Dr. Anurag Mangla.

This thesis would not have been possible without the Greek, Swiss, and international friends who contributed, each on their own way, to this work. I would like to say to all of you, a strong "thank you"!

I would also like to thank my partner Evangelia for her constant encouragement and support to address my concerns, anxieties, and frustration that a Ph.D. journey brings. Thank you for making this thesis possible.

Finally, I would like to express my deepest gratitude to my family, my sister Anastasia and my parents Konstantinos and Chrysoula who have supported me, despite the distance. Their love and guidance are with me in whatever I pursue.

Lausanne, 28 July 2017 G.L.

Abstract

Recent years has seen profound changes in building technologies both in Europe and worldwide. With the emergence of Smart Grid and Smart City concepts, the Smart Building has attracted considerable attention and rapid development. The introduction of novel information and communication technologies (ICT) enables an optimized resource utilization while improving the building performance and occupants' satisfaction over a broad spectrum of operations.

However, literature and industry have drawn attention to certain barriers and challenges that inhibit its universal adoption. The Smart Building is a cyber-physical system, which as a whole is more than the sum of its parts. The heterogeneous combination of systems, processes, and practices requires a multidisciplinary research. This work proposes and validates a systems engineering approach to the investigation of the identified challenges and the development of a viable architecture for the future Smart Building.

Firstly, a data model for the building management system (BMS) enables a semantic abstraction of both the ICT and the building construction. A high-level application programming interface (API) facilitates the creation of generic management algorithms and external applications, independent from each Smart Building instance, promoting the intelligence portability and lowering the cost. Moreover, the proposed architecture ensures the scalability regardless of the occupant activities and the complexity of the optimization algorithms.

Secondly, a real-time message-oriented middleware, as a distributed embedded architecture within the building, empowers the interoperability of the ICT devices and networks and their integration into the BMS. The middleware scales to any building construction regardless of the devices' performance and connectivity limitations, while a secure architecture ensures the integrity of data and operations. An extensive performance and energy efficiency study validates the proposed design.

A "building-in-the-loop" emulation system, based on discrete-event simulation, virtualizes the Smart Building elements (e.g., loads, storage, generation, sensors, actuators, users, etc.). The high integration with the message-oriented middleware keeps the BMS agnostic to the virtual nature of the emulated instances. Its cooperative multitasking and immerse parallelism allow the concurrent emulation of hundreds of elements in real time. The virtualization facilitates the development of energy management strategies and financial viability studies on the exact building and occupant activities without a prior investment in the necessary infrastructure.

Abstract

This work concludes with a holistic system evaluation using a case study of a university building as a practical retrofitting estimation. It illustrates the system deployment, and highlights how a currently under development energy management system utilizes the BMS and its data analytics for demand-side management applications.

Key words: smart building, intelligent building, systems thinking, scalable architectures, energy management, building management systems, building data model, real-time architectures, distributed computing, message-oriented middleware, ICT interoperability architectures, discrete event system, parallel architectures, building emulation

Résumé

Ces dernières années ont connu d'importants changements dans les domaines liés aux bâtiments, aussi bien en Europe que dans le monde entier. L'émergence des concepts tels que le réseau électrique intelligent, de la ville intelligente et du bâtiment intelligent a attiré une attention considérable et, par conséquent, a mené à un développement rapide. Le développement des nouvelles technologies de l'information et de la communication (TIC) permet progressivement une utilisation optimisée des ressources tout en améliorant le rendement énergétique du bâtiment et la satisfaction des occupants sur un large éventail d'opérations.

Cependant, la littérature et l'industrie ont mis en évidence certains obstacles et défis, qui empêchent leur adoption universelle. Le bâtiment intelligent est un système cyber-physique qui, dans son ensemble, représente une entité plus vaste que la somme de ses parties. La combinaison hétérogène de systèmes, d'algorithmes et d'interactions le constituant nécessite une recherche multidisciplinaire approfondie. Ce travail propose et valide une approche d'ingénierie des systèmes visant à répondre aux défis identifiés ainsi que le développement d'une architecture viable pour le bâtiment intelligent du futur.

Tout d'abord, un modèle de données pour le système de gestion de bâtiments (BMS) a été développé afin d'abstraire la sémantique des TIC et la structure géométrique du bâtiment. Une interface de programmation d'applications de haut niveau (API) facilite la création d'algorithmes génériques de gestion de ressources et d'applications externes. Ces dernières sont ainsi indépendantes de toute instance du bâtiment intelligent, favorisant la portabilité de l'intelligence et réduisant de surcroît le coût. En outre, la répartition de charge et la virtualisation assurent l'élasticité du système, indépendamment des activités des occupants et de la complexité des algorithmes d'optimisation.

Ensuite, un middleware temps-réel et orienté message assure l'interopérabilité des périphériques, des réseaux TIC et leur intégration après du BMS. Son architecture embarquée et distribuée au sein du bâtiment lui confère de nombreuses propriétés indispensables au bâtiment intelligent du futur. Le middleware s'adapte à n'importe quelle structure de bâtiment, indépendamment des performances des périphériques et des limitations de connectivité, tandis qu'une architecture sécurisée garantit l'intégrité des données et des opérations. Une étude approfondie de l'efficacité énergétique et la performance du BMS valide la conception proposée.

Résumé

Un système d'émulation de « bâtiment-dans-la-boucle » virtualise les éléments dominants du bâtiment intelligent (par exemple, charges, stockage, génération, capteurs, actionneurs, utilisateurs, etc.). L'émulateur, basé sur une simulation d'événements discrets, est directement intégré dans le middleware, garantissant de la sorte une abstraction de la nature virtuelle des instances émulées vis-à-vis du BMS. Les entités virtuelles évoluent dans un environnement multitâche coopératif et son important parallélisme permet l'émulation simultanée de centaines d'éléments en temps réel. Cette virtualisation facilite l'élaboration de stratégies de gestion de l'énergie et l'étude de viabilité financière en synergie avec le building réel et ses occupants, évitant ainsi un investissement dans l'infrastructure nécessaire à ces études.

Ce travail se termine par une évaluation du système holistique appliqué à l'étude d'un bâtiment universitaire, sous forme d'une estimation réaliste de réaménagement. Elle illustre le déploiement du système et démontre comment un système de gestion de l'énergie pourrait tirer profit du BMS et de sa capacité de traitement de données, dans le but d'installer une plateforme de gestion de la demande en énergie du bâtiment.

Mots clefs : bâtiment intelligent, approche systèmes, architectures évolutives, gestion de l'énergie, systèmes de gestion de bâtiments, modélisation des données du bâtiment, architecture temps-réel, calcul distribué, middleware orienté message, architectures d'interopérabilité TIC, système d'événements discrets, architectures parallèles, émulateur de bâtiments

Contents

List of Figures

List of Tables

List of algorithms

Introduction

More than half of the world, and 74% and 82% of the European and North American populations respectively, lives in urban areas [1]. The trend of increasing urbanization is expected to continue in the following decades. Thus, the sustainable development and the wellbeing enhancement of urban centers are of paramount importance.

The emergence of advanced information and communication technologies (ICT) have led to the introduction of technologies such as the Internet of Things (IoT) and big data analytics, which enable real-time monitoring and control, improved services, and efficient decision-making. In this context, the Smart City is the vision for integrating the ICT and city operations, services, and infrastructure for addressing urban challenges. Over the years, the concept has been applied to several areas, such as energy, water, waste management, mobility, public safety, and critical infrastructure monitoring. Specifically, the initiation of the Smart Grid (SG) vision made energy management, in particular, the key driver for the Smart City.

Traditionally, energy has been produced in centralized power plants, transmitted, and then distributed to the cities' residential, commercial, and industrial consumers. However, there is now a shift from centralized to decentralized generation, which changes the energy landscape. Renewable resources, as well as energy storage systems, are increasingly being integrated into new and retrofitted buildings. In that sense, energy consumers are becoming energy "prosumers". Furthermore, residential and tertiary sectors were responsible for around 43% of total final energy consumption in 2015, according to the EU Reference Scenario 2016 [2]. Therefore, the building domain plays a significant role in the energy policies of a city.

It is evident that building performance has been advancing continuously with regard to operating efficiency and occupant wellbeing. There are several identified drivers for such developments, which mostly revolve around increasing the value of the building [3]. This added value is not only financial. On the contrary, it encapsulates the performance, comfort, and overall satisfaction for its users. Additionally, there is a shift of interest towards quality and improved operational costs over the building life, rather than for the initial investment [4]. In fact, research has shown that nearly 80% of the energy usage in a construction's lifecycle is related to the operational stage [5]. Therefore, its longevity, the ability to maintain the value over an extended period and shifting conditions is a major building performance indicator.

For the last few decades, the so-called "intelligent building", denoted the conceptual representation of the future building. The ICT of the last decade have been a major driver of rapid growth and the realization of such a future building vision. The involved ICT are diverse; some have already been deployed and validated for years, while others are new but very promising. Nowadays, the ICT in a building grow beyond the building automation systems and devices like the sensors, actuators, and controllers. There is now a significant number of consumer-owned devices available, such as smartphones, digital home assistants, connected locks, CCTV, smart appliances, intelligent thermostats and lighting, etc.; those are commonly referred to as IoT. Those will not only provide new sources of data on human activities, but they will also provide greater granularity for action. Moreover, as the incentives are market, lifestyle, and wellbeing-oriented, they are more likely to commit to such purchases.

The Smart Building (SB) is, therefore, an evolution of the "intelligent building" concept that achieves significant value improvement [6] by leveraging the new software algorithms, the augmented data sources, the new energy generation and storage of the building, as well as the energy market progression. In that sense, the SB aims beyond the building automation scope. Practically, a SB is distinguished by the utilization of novel and consumer ICT, the high adaptability to changing conditions, improved energy management and sustainability, and occupant interaction and empowerment. Thus, regardless of the technologies and intelligence in place, a SB should continuously improve on the energy efficiency aspect without jeopardizing the perceived comfort and satisfaction of those within, while maintaining that over a long time.

Despite the advancements, there are still social, economic, and technological barriers and challenges that hinder the adoption of SB [7]. In the literature, there are already various solutions for addressing *individually* the challenges of, including but not limited to, interoperability, reliability, complexity, security, privacy, and cost. However, the SB has become a sophisticated heterogeneous cyber-physical system (CPS); the need for *multidisciplinary research* and *systems thinking* has recently become more evident. Undoubtedly, the intersection of ICT, energy, and occupants, which define the SB, are strong foundations for such multidisciplinary research. In fact, it is a necessity for understanding its complex interactions and the influence of its stakeholders.

> *A system is more than the sum of its parts. It may exhibit adaptive, dynamic, goalseeking, self-preserving, and sometimes evolutionary behavior.*
>
> *Many of the interconnections in systems operate through the flow of information. Information holds systems together and plays a great role in determining how they operate.*
>
> *The least obvious part of the system, its function or purpose, is often the most crucial determinant of the system's behavior.*
>
> Donella H. Meadows [8]

This work proposes and validates a systems engineering approach to the investigation of the identified challenges and the development of a viable system architecture for the future SB. Throughout the chapters of this dissertation, the reader will progress through a series of ideas, technologies, architectures, and implementations that seek to answer both wide- and narrow-scope social, technical, and research challenges in a sustainable SB system.

Dissertation Outline

This dissertation is divided into six chapters. Each chapter investigates a discrete aspect of the SB system design. Chapters 2 - 4 are the three major contributions of this work in the form of SB subsystems for addressing the challenges identified in Chapter 1. Finally, Chapter 5 concludes and validates the system design with its deployment in a university building as a case study.

Chapter 1 discusses the concept of intelligent building, its stakeholders, and major opportunities. The terms of building automation and SB are frequently used interchangeably. Thus, this chapter presents the current building automation solutions and compares those with the concept of SB, highlighting the advantages of the latter. Moreover, the challenges and barriers for SB adoption are investigated. Finally, several stakeholders are identified, and the chapter concludes with two prominent SB applications.

Chapter 2 focuses on the core component of any SB, the building management system (BMS). It presents a model-based approach to the design of the BMS, which enables a semantic abstraction. A matching application programming interface (API) is designed by the identified requirements and facilitates the creation of generic algorithms and applications regardless of the particular building and ICT characteristics. An event-driven architecture ensures the near real-time operation, eliminating latency introduction by the BMS. Moreover, the load balancing ensures the scalability regardless of the occupant activities and the complexity of the optimization algorithms. Two case studies, an occupant localization system, and a thermal simulator leverage the BMS-exposed abstractions and API in order to provide high-level data services, validating the extensibility of the model-based BMS design.

Chapter 3 addresses the major challenge of interoperability and technology fragmentation which creates social, financial, and technological barriers as identified in Chapter 1. The chapter proposes a real-time, message-oriented middleware (MoM) system as a distributed embedded architecture within the building. This system addresses the challenges of extendibility, scalability, adaptability, and security of the ICT systems and integrates them into the BMS. An object-oriented programming paradigm and a layered architecture for each distributed middleware node ensures

the expandability in supporting new ICT devices and protocols. A case study on several platforms, as distributed nodes, investigates the performance and energy efficiency of such middleware design for SB.

Chapter 4 extends the previous chapter by introducing the real-time virtual middleware concept. This is a discrete-event simulation scheme as a "building-in-the-loop" emulation system, which "virtualizes" common SB elements such as loads, storage, generation, sensors, actuators, users, etc. As the virtual middleware is an extension of the physical one, the BMS remains agnostic to the virtual nature of the emulated instances. Thanks to a cooperative multitasking design, hundreds of building elements are emulated concurrently and in real time. Such a system permits the validation and optimization of several algorithms in the actual building without a prior investment in the necessary infrastructure.

Chapter 5 evaluates the proposed SB system as a whole, using a case study of a university building, as a practical retrofitting assessment. Moreover, it highlights how an energy management system in development within the research group, leverages the BMS abstractions and internal data analytics for demand-side management.

Chapter 6 concludes this work, highlights its overall contribution, and suggests possible future work for improving and extending the proposed architecture.

Research Contributions

This dissertation wishes to provide a system architecture that mitigates some of the barriers in the SB adoption, cf. Chapter 1. By addressing the challenges, it aspires to catalyze the public SB adoption as a necessity towards the Smart City and Smart Grid collaborative environments. The major original contributions of this dissertation are listed below.

- **Systemic approach** to the intelligent building and **identification of the challenges and barriers** that hinder the adoption.
- Development of a **model-based BMS** as a highly adaptable management system.
- **Semantic abstractions** in the BMS for reduced complexity and decoupled external algorithm development.
- Proposal of a flexible load balancing architecture for BMS for facilitated **scalability** regardless of the algorithmic complexity, building size, ICT infrastructure, and occupant activities.
- An **event-driven real-time server** ensures the low latency requirements without impacting the functionality.

- Novel **distributed and scalable message-oriented middleware** system for SB, **adaptable** to any building construction, ICT topologies and capabilities.
- **Layered middleware node architecture** for easy extensibility of the functionality and supported ICT.
- Middleware optimized and validated on **embedded hardware**, reducing the required investment and energy to run it, while minimizing its visual intrusiveness.
- Middleware augmented with a **secure architecture** ensuring the data and operations integrity.
- Innovative **SB emulation system** based on discrete-event simulation, supporting most of the contemporary SB elements.
- The real-time operation in parallel with the physical devices and the integration with the message-oriented middleware makes the **BMS agnostic to the virtual nature of the emulated elements**.
- This **"building-in-the-loop"** emulation tool that can be used for evaluation of energy management strategies and financial viability studies without a prior investment in the necessary and costly infrastructure.
- A **case study on a physical university building** highlights the practical deployment of the proposed SB architecture for energy management practices.

1 Smart Building Perspective

This chapter assesses the potential of advanced technologies in buildings for the transition towards the Smart Building (SB) era. In fact, nowadays the understanding of an intelligent building goes beyond automation. The SB is defined as the orchestration of policies, stakeholders, and novel systems that challenge the traditional practices of building automation. Such systems are highly heterogeneous due to, among other things, the building, the investment size, and the current market trends. Some have long-term validated value and results, while others are only newly introduced but highly promising. This chapter develops and defines the SB concept and highlights its benefits compared to traditional automation. Moreover, it explores the barriers that hinder its adoption and highlights the challenges in the state of the art, some of which are addressed in the following chapters. The chapter concludes with the major SB stakeholders and applications that leverage its advantages.

1.1 Building Automation

Building automation systems (BAS) for the monitoring and controlling of building environments are becoming a standard consideration. They are aimed mainly at energy efficiency through heating, cooling, ventilation and lighting control.

One can categorize the building automation standards based on their primary domain of functionality [9]. There are generally three hierarchical levels of functionality in a given BAS. The *management* level is where all the information from the entire system is collected, aggregated and represented in a unified way to the operator. This is where the different control and management decisions are introduced, either by the operators or by an automated optimization agent. The long-term data storage, analytics and performance reports are also generated at this level. On the other hand, the *automation* level includes all the infrastructure capable of applying a predefined scenario or maintaining a control set point. On this level, the automation infrastructure acts as a delegate of the *management* level to the end devices. This can be in the form of a sensor values accumulator, control dispatching, data pre-processing, alarm triggering, etc. Finally, on the *field* level are all the end devices, communication networks and in general the infrastructures that interface with the physical environment of the building.

Some prominent BAS are the following:

- **BACnet's** popularity has grown over the years and it has become the leading technology in building automation. It is frequently marketed as the universal building management and automation standard. It provides the means to manage the building regardless of its construction characteristics. Specifically, its object-oriented programming (OOP) approach standardizes the representation of data and processes within the building. However, it does not define the internal data structures, control logic and configuration vectors for each field device. Those are left open to their respective manufacturers. Similarly, neither their data-link nor their physical layers of communication networks are standardized by the BACnet protocol. Such networks are simply interfaced on the BACnet's network layer. This ensures the highest possible interoperability between the various vendors and heterogeneous devices.
- **LonWorks**, on the other hand, standardizes the automation and field levels in a BAS. It is based on the LonTalk communication protocol, a dedicated system-on-a-chip (Neuron Chip) together with the necessary network management utilities and infrastructure. The Neuron Chip contains the entire LonTalk protocol stack, as well as the required firmware and operating system for participation in the LonWorks network. Moreover, the standard does not require a particular network topology; repeaters, bridges, routers, and gateways enable peer-to-peer and direct connections between the field devices.
- **KNX** standardizes, in a similar manner to LonWorks, both the automation and the field levels. It is the successor of three successful European standards; the EIB, the BatiBus, and the EHS. On the field level, the standard defines the physical layer that operates over twisted copper pairs, power line communication (PLC), radio frequency or Ethernet.

The BACnet and LonWorks systems have achieved considerable adoption worldwide, while the KNX has a strong European market presence. However, even if the automation market has gone a long way towards standardization nowadays, such protocols are still focusing primarily on the automation aspect; however, the Smart Building (SB) notion extends far beyond that.

1.2 Smart Building Perspective

For the past few decades, any conceptual design and proposal representing the future building was labeled as a smart or intelligent building. Frequently, the terms "smart" and "intelligent" have been used interchangeably. Together with building automation, they create ambiguity in their interpretation and understanding for both clients and engineers/researchers.

Buckman [3] and Ghaffarianhoseini [10] discuss this issue, explore the literature for smart/intelligent building interpretations, and extract the common features leading to a generic definition. Moreover, the authors of [11] address the ambiguity in smart infrastructure as enabling technologies. In this study, the authors differentiate between smart and intelligent infrastructure, as according to them, the former collects and processes the data into actionable decisions while the latter augments that with autonomous and dynamic adaption to changing conditions. According to their state of the art study, there is a significant increase in the use of the term "smart" in place of "intelligent" in recent publications, a shift that can also be associated with the wide use of the term for heterogeneous technologies, e.g., from Smart City and Smart Grid (SG) to smartphones.

There are four regions of innovation and performance evaluation which are relevant to SB, according to [3]. The continuous and concurrent development of those regions highlights the differentiating aspects of the SB compared to current building automation technologies.

1. **Construction**: the building's physical shape and materials for improving its performance. E.g., improved isolation, phase-changing walls, etc.
2. **Control**: the process for implementing the intelligence decisions and interacting with occupants and building environment. These range from a thermostat to a more advanced BAS maintaining the internal comfort settings.
3. **Intelligence**: the methods for collecting, analyzing, and responding to incoming data in order to meet the comfort and energy priorities. E.g., an SB that controls the heating based on external and internal temperature readings, as well as occupant presence.
4. **Enterprise**: the augmented methods for creating higher dimensionality data, leveraging the integration of intelligence with external data sources and systems, in addition to the processes for leveraging those data in order to improve both energy effectiveness and comfort accommodation.

As already mention in the introduction, it is the information and communication technology (ICT) systems, both as software algorithms and Internet of Things (IoT) devices, that enable

the SB to achieve this significant advantage over common building automation. In fact, the introduction of advanced ICT solutions, more recently by the IoT, the adaptability to changing occupant behaviors and their excellent integration, as well as the emergence of big data and artificial intelligence techniques, have recaptured the public interest in the domain of intelligent buildings.

Furthermore, the adoption of such a high number of physical environment sensing and acting infrastructure has enabled creative applications to materialize. This is where occupants, their personal devices, and their activities get integrated into this SB ecosystem with the ultimate desire of a more sustainable future while achieving a higher living standard. For example, literature includes a number of implementations where humans get involved in a continuously increasing energy and carbon footprint [12, 13, 14, 15, 16, 17].

Apart from the social benefits, the plethora of data sources will enable new business models as well advancing the management market beyond the means of automation. Such opportunities could focus, for example, on the data analytics driven by the numerous physical environment sensing devices. In fact, adaptability to occupants' behavior changes and preferences has been largely neglected in current building automation systems. Such business opportunities will not only revitalize the slightly stagnated building automation market, they will likewise encourage and fortify the opportunity for new enterprises to enter this market.

It is clear that the SB consists of a rather wide scope concept; the effort to provide a universal and holistic definition for it is a non-trivial endeavor. What is certain though is that the SB is a large evolution in building automation with significant value introduction to the building market [6]. A number of researchers have attempted to give a comprehensive definition of the SB. In the author's opinion, the definitions given by Prof. Clements-Croome and Dr. Al Waer are the most appropriate ones.

> *An intelligent building is one that is responsive to the requirements of occupants, organisations and society. It is sustainable in terms of energy and water consumptions besides being lowly polluting in terms of emissions and waste: healthy in terms of wellbeing for the people living and working within it; and functional according to the user needs.*
>
> Derek Clements-Croome [18]

> *A sustainable intelligent building can be understood to be a complex system of inter-related three basic issues People (owners; occupants, users, etc.); Products (materials; fabric; structure; facilities; equipments; automation and controls; services); and Processes (maintenance; performance evaluation; facilities management) and the inter-relationships between these issues.*
>
> Husam Al Waer [19]

The author of this thesis, after extensive state of the art review and brainstorming, concluded to the visualization and definition of the SB as seen in Fig. 1.1. In the author's opinion, such visualization facilitates the understanding of the SB placement and interaction with other relevant research domains.

Therefore, according to the figure, the SB is the ultimate aggregation of three major factors or influence poles: the ICT, the occupant, and the physical building construction. All of them are necessary for its existence, and each one of them are differentiating and contributing to the SB's value.

Particularly, what is unique in the conception of such a visualization is its design in the form of a Venn diagram. Such a scheme effectively highlights the logical relations between the three different poles, or sets for a Venn diagram. Each set, similar to a Venn diagram, consists of a finite collection of elements or technologies relevant to that pole.

To begin with, the ICT set includes all the "smart" infrastructure and devices. However, there exist a finite number of such devices that serve in particular the needs of building management. The finite number of ICT devices that serve this purpose can be expressed as the intersection of ICT and building regions in the Venn diagram of Fig. 1.1. This intersection is in fact the well-understood building automation technologies and systems.

As seen in Fig. 1.1, this set does not include the element of occupant. The latter is in the intersection of the ICT and occupant regions. The corresponding set indicates the advanced ambient intelligence and consumer-oriented solutions. These systems' scope is not necessarily constrained to the building setting, as they include, for example, smart transportation, wearables and other cloud services.

The last intersection of occupant and building regions indicates passive energy consumption optimization solutions. Those include the more energy-efficient construction materials, the building sustainability policies, strategies for energy use reduction or even energy awareness and occupant empowerment and integration.

Ultimately, the SB can be described by the final intersection of the regions generated by the three previous intersections: the building automation, the ambient intelligence, and the energy consumption, cf. Fig. 1.1. As such, the SB is the collection of technologies and elements that originate and borrow the characteristics of all three distinct regions.

Finally, the arrows in the figure illustrate their symbiosis and interaction. As such, the occupant is necessary for financing the investments in ICT infrastructure, which in turn manage the building according to the defined targets and requirements. In the end, it is the building that supports and enhances the occupants' living and wellbeing.

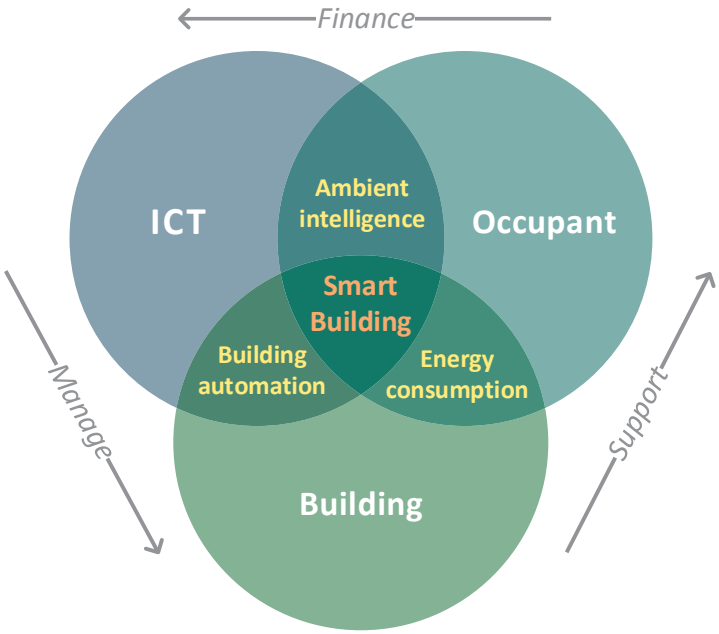


Figure 1.1 – The Smart Building concept visualization

1.3 Barriers to Adoption

Despite the advancements in the state of the art and the improved value to the building sector, there are still social, economic, and technological barriers that hinder the adoption of SB [7]. Balta-Ozkan [20] explores the social barriers to adoption of SB with literature, experts, and public opinion assessments. Although an exhaustive study is beyond the scope of this thesis, the following subsections cluster and present the most prominent difficulties in smart infrastructure adoption as identified by the author.

1.3.1 Interoperability

One cannot expect a single manufacturer to provide continuous product development and support. Thus, the only viable approach to reassure the market is the existence of compatible products from multiple manufacturers.

However, in light of the quickly advancing SB market, major building automation firms and start-ups alike were quick to introduce solutions as part of their proprietary standards. While this enabled a quick capture of business opportunities as more parties introduce their proprietary implementations, it started to become a Tower of Babel where hardly any integration between existing solutions was possible.

Engineers identified the interoperability concern over a decade ago, which luckily current building automation systems have resolved to a degree. BACnet Manufacturers Association, for example, claims that more than 800 unique vendors globally use the standard with an

increasing trend. The same belief is shared with the LonWorks standard group, claiming a 4000-product range and their devotion to the open standards. As a matter of fact, the literature even demonstrates designs for multi-protocol devices [21], eliminating the need for specialized gateways for inter-protocol communication, thus increasing the potential product range available from each manufacturer. However, the interoperability of current standards comes at a not-so-evident cost. The fact that the major automation standards are open does not imply they are offered for free. For example, Echelon®, who governs the LonWorks standard, requires royalty fees for every device using their Neuron Chip. BACnet International, on the other hand, does not charge a fee; however, a yearly compatibility certification is necessary [22]. Similarly, for the KNX standard, while there is not a per device fee, the necessary configuration tool (ETS4) requires a license [23]. All those fees for the small device vendors can be a costly exercise. Unfortunately, this becomes even more prominent when vendors prefer to develop a base product on their proprietary protocol and to charge in addition for the inclusion of a standard interface; thus, by trying to get a market advantage against big players, they further fuel its fragmentation [24].

All in all, it is apparent that the integration of different and sometimes even the same standard devices made by various vendors is not always a trivial task. There is a high discrepancy between the interoperability that they are supposed to have and the actual one. Therefore, the adoption of universal connectivity standards, as well as the development of interoperability and integration technologies are critical to overcoming this barrier. Chapter 3 presents such interoperability enabling technology without impact on functionality, cost, or performance.

1.3.2 Security and privacy

Security and privacy are frequently mentioned as one of the top concerns in SB and smart infrastructure in general; thus, they are important factors that influence the adoption of SB technologies. It is therefore imperative to understand the challenges and evaluate the benefits of an IoT-enabled building. As the IoT devices are communicating over public networks and well beyond the building environment, there is an increased risk of data compromise. In the literature, there are several attempts to identify these security and privacy challenges. For example, Roman and Sicari [25, 26] identified various issues that need to be addressed before the security and privacy requirements are met.

As a matter of fact, the term of security defines multiple aspects such as integrity, confidentiality, authentication, authorization, non-repudiation, and availability. Moreover, as the SB is a cyber-physical system (CPS), the security is not only applicable to data. On the contrary, it is important to understand that a potential security compromise could jeopardize the building's physical safety as well. Nevertheless, security is seen by some experts as a technical problem that can be addressed by state of the art solutions [27, 28].

Privacy, on the other hand, is a more challenging barrier, as it is highly related to the policies and legislation in place [29]. Privacy of an SB system requires in general the protection of

occupants' private data, patterns, and interactions with other people or objects. Unfortunately, such privacy challenges in IoT and concerns in big data analytics are only partially addressed. However, there is an ongoing effort to address those. For example, practical solutions such as cluster-based anonymization schemes [30] could provide the desired privacy to the individual without affecting the data analytics benefits and business models.

1.3.3 Financial

Concerns have been also raised by various stakeholders over the high cost of the SB technologies [20]. Firstly, the purchase and installation cost is a considerable expense that may not be justified by low to mid-income households and short-term tenants. Secondly, the smart infrastructure may require specialized maintenance and repairs that are perceived as costly and a potential financial risk. Moreover, the market fragmentation, the planned obsolescence, and the lack of interoperability between generations of technologies require frequent and considerable upgrades which exacerbate the situation.

However, as nearly 80% of a building's energy use takes place over its operational stage [5], there are indeed financial benefits from reduced and optimized energy use. Despite the generally low cost of electricity which could make such systems less attractive, the demand side management (DSM) with the energy providers could trigger additional incentives through smart energy contracts. Moreover, depending on the policies and development plans in place, the governance can also subsidize SB technologies for improved urban energy management and efficiency. However, those are strongly related to the SB's final performance delivery, which is not always the case, as seen in the next subsection.

1.3.4 Performance distrust

This trust barrier mainly relates to the modern SB systems, as their IoT infrastructure is considered by same as a market trend. This can be blamed on the technical gap between the promised and the delivered solution's performance. In fact, poorly implemented systems may quickly become redundant and unused.

Furthermore, there is still not enough comprehensive research on the profitability and performance of modern IoT-based SB systems. On the other hand, the majority of the literature work, reviews, and long-term evaluations have been conducted on the traditional BAS [31]. This can inhibit industrial and other large-scale projects' interest in novel SB systems, as the benefits cannot be easily estimated. Legacy automation standard supporters strongly emphasize the verified performance of their ecosystems. These systems are much more mature, and they have been installed and evaluated in varying premises and schemes, highlighting their effectiveness through multiple case studies. Throughout the years, many research groups used those standards as the sole means of automation and energy management evaluation. Those research studies increase the impact and credibility of such

standards. Furthermore, plenty of their enterprise customers prefer the certified devices and value the deliberately slow process by which these standards evolve and are refined. The aftersales service also ranks high on the priorities of the B2B relations, in which the legacy building automation vendors generally excel.

It is true that the IoT-based system can be somewhat overmarketed nowadays. In fact, Gartner, a technology research and advisory firm, ranked them at the peak of their inflated expectations in their Hype Cycle for two consecutive years [32, 33]. The primary concern is the lack of clear added value for the end consumer, which usually considers them more of a lifestyle gadget. To make matters worse, many products are named and marketed as IoT when they are just smartphone-connected devices without demonstrating any of the novel IoT advantages. Additionally, many building automation specialists believe that there are still not enough models and algorithms for utilizing the enormous number of data sources generated by the extensive connectivity and ubiquitous computing of IoT.

However, the research in recent years on IoT and data science has paved the way for new business models based entirely on the building's sensors data analytics. Thus, it could take years for the IoT to solidify, but it will eventually introduce the innovating concepts that would revolutionize public regard of the building and its energy use. The literature, in fact, proposes various behavioral analysis solutions for energy recommendations combined with automation. Thanks to that, the potential scenarios for financial and comfort returns will increase further in scale. By the end of the day, IoT potential in SB will be demonstrated by the risk-taking companies that are willing to develop novel products and data services.

1.3.5 Reliability

Due to the intrusive nature of human activities, a key concern for the SB is reliability. Reliability is different from the performance barrier, as the latter describes how well the system meets the needs, demands, and preferences, while the former defines the quality of being trustworthy or performing consistently. Some measures of reliability are, for example, what happens if things go wrong, and how likely those things are to go wrong. In fact, reliability is a good measure of the unacceptable, sporadic, stochastic behavior of a system that interferes with the desired outcomes. In particular, a network of things needs to be adaptive and resilient to communication errors by providing failsafe mechanisms for information distribution. Moreover, reliability is defined also by its tolerance not only of internal issues but also of potential configuration faults by the users.

While reliability is highlighted as one of the major concerns and a deterring factor for SB solutions [20], consumer-grade systems are not addressing it effectively [34], leading to frustration and distrust. Nevertheless, some researchers have already recommended some reliability-enhancing schemes for the smart infrastructure [35, 36] likely to be found in future SB.

1.3.6 Adaptability

Buildings, especially large ones, are not static systems. On the contrary, they are dynamically evolving through their lifecycle based on their occupants' patterns, behaviors, and preferences. Any process that aims to adjust and influence the building environment, such as by means of automation, is intrusive by its nature and can potentially interfere with the occupants. This can be more pronounced, if for example, their preferences and patterns have diverged over time from the initial system configuration. As such, adaptability describes the performance of an SB to anticipate and self-adapt to such changes.

However, while the literature has proposed several schemes for tracking and modeling human behavior in order to adapt accordingly, commercial products have yet to catch up. Most of the BAS that have been offered up until now have generally been reactive to changes rather than adaptive. Therefore, future SB systems need to have adaptability to changing conditions as one of their design principles in order to differentiate from currently marketed systems.

1.3.7 Building diversity and multi-stakeholder environment

Buildings are not only dynamic but also very diversified systems. Practically, unless they are part of a specific urban and architectural design, they are unique in aspects such as size and architecture, materials, users, climate, energy generation and storage infrastructure, and cabling, as well as heating capacity and isolation.

Therefore, introducing any form of intelligence would require a management system to be specifically designed for that particular building. This is highly inefficient financially due to the necessary dedicated design and configuration work hours. Such a workload increases the overall cost of the installation and deters potential investments in such technologies. Furthermore, such an approach can impact the reliability as well, as each installation is a newly designed and realized system with much more limited validation compared to a universal adaptable design.

However, the literature mainly focuses on the interoperability of the ICT systems rather than on solutions to address building diversity. This could be justified by the fact that the ICT's scope is beyond building management, when in essence it does not always encounter the physical environment diversity barrier. Nevertheless, this work proposes a distributed middleware solution, cf. Chapter 3, that can mitigate the impact of building diversity on the design of SB systems.

1.3.8 Complexity

Another considerable barrier to the adoption of SB lies in their complexity, not only during the installation but also in operation. As a third party cannot fully anticipate the specifics of an individual's needs, frequently the user of a SB system would need to have some level

of expertise in the system's operation and management. As the individual may not be able to adapt to complex systems with frequently difficult interfaces, the performance of the system would suffer considerably with switching living patterns and priorities. Moreover, simply collecting and displaying environmental data does not help either; it can even lead to information fatigue, particularly in elderly individuals. However, an SB targets internal quality of life improvement rather than the complication of it. Thus, one of the primary drivers and a barrier to overcome for the SB is its flexibility to changing occupant patterns and ease of use with engaging interfaces and a high abstraction level of data visualization.

1.4 Stakeholders

Generally, business models and markets are driven by the user-perceived value in a product. However, unlike traditional solutions, who we define as a user greatly varies in SBs. As a matter of fact, the entities interacting with SB systems are not only the users or occupants. Throughout its long lifecycle, the SB needs to be designed, tested, managed, repaired, and even upgraded. These activities involve a number of parties or stakeholders with financing models providing the incentives and the ICT tools supporting the objectives' achievements.

> A *stakeholder* in a system is an individual, group, or organization, having an interest and an influence in the design, realization and operation lifecycle phases of the system.

As a system of systems, the SB is a multi-stakeholder environment with competing powers and interests. As the number of them increases, their relationships grow exponentially too. Incorporating them into the SB means understanding and managing their different roles and purposes in order to achieve collective goals through collaboration and shared interests. In fact, a system designed to consider its stakeholders shifts the focus from individual stakeholder addressing to balancing their relationships optimally.

However, a thorough and holistic stakeholder analysis is beyond the scope of this thesis, as it requires a systematic gathering and analysis of qualitative information in order to cluster, determine and prioritize interests and influences. Nevertheless, the SB system design process has taken the stakeholders' influence into consideration explicitly and implicitly in order to identify the requirements and state-of-the-art shortcomings. The following stakeholders, or clusters of them, are some of the most prominent ones in the SB ecosystem.

Occupants are influenced by the SB not only in terms of comfort, but also design aesthetics, ambiance, and intrusiveness. This stakeholder controls a large financial aspect of the SB through consumer participation and interest. Inevitably, it plays the largest role in the SB's acceptance and success. Occupants prioritize monetary savings, improved comfort, convenience, security, and for some, the green ecological footprint. The occupants are also very sensitive to perceived reliability, operational complexity, and privacy [20, 37, 34].

Utilities and energy providers consist of the second largest stakeholder when considering the SB's scope within an urban energy system. The SB is essential to optimizing energy use, generation and storage; thus, the energy services offered by the SB systems directly influence the available business models for the utilities.

Governance and policy-makers are also a significant stakeholder nowadays due to Smart City initiatives. For this stakeholder, the SB can support not only the energy but also water and mobility management. Additionally, the policy makers have the power to recommend CO_2 targets, and energy and water strategies, while the governance can enforce the regulation and provide research funding and subsidies for building retrofitting. All those actions can greatly influence both the overall SB market momentum as well as specific areas of applications.

ICT providers are supplying the solutions for transforming the SB. Those can be anything from electronics, automation and network infrastructure to software and cloud services. While this stakeholder's interests are based mainly on financial models, its influence varies greatly depending on its market momentum. Most importantly, it is frequently the source of the technology fragmentation in the SB ecosystem.

Building owners and managers consist of another fairly critical stakeholder group, especially during the initial design and retrofitting phase. Unlike tenants, this group has the final control over the decision to invest in a SB system. Unlike an occupant, it is primarily driven by the monetary gains through efficiency, energy optimization, remote management and building value improvement. Moreover, an owner may or may not also be an occupant stakeholder; however, those two stakeholders are not equivalent. The latter focuses on short to medium-term convenience while the former looks for long-term improvements.

Standard makers consist of a yet another diversified stakeholder. They establish the standards for operation, monitoring, and interoperability in ICT and between stakeholders. The stakeholder focus is not necessary limited to SB. The standard makers have the capacity and interest to resolve some of the social and industrial barriers in SB adoption.

Building designers include anyone involved in the construction of a SB, from architects and civil engineers to SB system designers.

Researchers are crucial stakeholders that drive innovation in SB environment. Through their work and collaboration, new advanced technologies, algorithms, simulation tools and optimization approaches continuously transform the SB space.

1.5 Applications

While the stakeholders' groups are rather numerous, the SB applications can be clustered mainly into two groups: firstly, the DSM and the integration of SB in a SG, and secondly, the user engagement through the integration of people, processes, and products.

1.5.1 Demand side management

DSM refers to the processes and measures to improve energy delivery on the consumption side. Such processes can be clustered into two groups. The first group's solutions aim for *energy efficiency* and possibly zero-energy buildings through optimized energy use. The second group's solutions have a wider scope; they integrate and control in real time the renewable energy generation and storage, as well as actively participate in regulating the power exchange with the SG using demand response (DR). Palensky [38] provides an overview of the domain and classifies the DSM practices and processes.

There are two major approaches to energy efficiency in a building. The most common one is the *passive* approach, which focuses on the improvement of the building's thermal envelope by incorporating improved isolation and thermal storage material (e.g., phase changing). Secondly, it promotes the wide adoption of energy-efficient appliances through sensitization campaigns. The second approach to energy efficiency calls for *active* building automation solutions, which is the most relevant to this thesis.

Furthermore, there is an increase of distributed, renewable energy sources and storage integrated into the new and retrofitted buildings. Moreover, the DR programs enable an active change in energy usage by the end customer in response to price changes, incentive payments, or signals from the energy system operator. Nevertheless, there are still challenges and barriers to the deployment of such programs as scrutinized in [39, 40, 41].

The available DR programs vary with region and system operator. Han [42] categorizes them as follows:

1. **Incentive-based**
   (a) Direct load control (DLC): the system operator remotely controls the consumer's electrical equipment on short notice in exchange for an incentive payment.
   (b) Interruptible/curtailable rates: the consumer receives a discounted rate in exchange for cooperative load reduction during system contingencies. If the consumer fails to comply, they can be penalized.
   (c) Emergency DR: incentive payments for a reduced load in response to emergency signals.
   (d) Capacity market: consumers agree to provide pre-defined load reduction when the grid is in need in exchange for guaranteed payments.
   (e) Demand bidding programs: large customers can bid for curtailing at specific prices.
2. **Time-based rates**
   (a) Time-of-use (TOU): these policies financially penalize certain periods of time in order to discourage the use of energy during those periods. As the financial penalties are usually defined in the contract, there are rarely changed to match energy generation.

(b) Critical peak pricing (CPP): similar to TOU but less deterministic, as CPP events can be triggered by system contingencies. However, the rates are typically pre-defined and CPP events are not continuous.

(c) Real-time pricing (RTP): those rates vary continuously during the day as the wholesale market energy prices are reflected. While they are not predetermined like the CPP, the rates can be communicated ahead of time (e.g., a day ahead) so the consumer can adapt their energy use accordingly.

1.5.2 User engagement

The definition of a user is not limited solely to the occupant. In fact, the term user denotes any stakeholder that utilizes an SB system, e.g., occupant, managing firm, owner, etc. Hence, comfort enhancement denotes accordingly the ease of doing certain building related activities, such as management, optimization, monitoring, access control, etc.

Energy consumption awareness and recommendations for action can effectively motivate energy-efficient behavior [12, 13]. Thus, efficiency gains can be realized by inducing behavioral changes in people through appropriate feedback on their energy consumption. Studies have shown that it needs to be frequent and over a long time, to use interactive elements, and to be presented in an appealing way to the occupant [43]. Moreover, feedback in environmental units and high-level abstraction, e.g., a number of trees to offset the CO_2 emissions associated with the consumption, resulted in a greater reduction compared to direct feedback in energy units (kWh) [44]. Furthermore, studies have shown that social influence and comparative feedback has an even greater potential for energy-efficient behavior [14, 15, 16, 17]. However, some authors have raised concerns as to whether energy consumption feedback with existing in-home-displays leads to the desired energy reduction, and highlighted the necessity for novel feedback devices for user engagement and long-term studies [45, 46, 47].

Comfort enhancement without increased energy consumption is the most distinguishing feature of the future SB, thanks to their increased adaptability and IoT. As occupants' behavior has been shown to greatly influence energy consumption, there is an ongoing effort in modeling their complex and fairly stochastic behavior [48, 49]. In fact, their activities and behavior are the most important input for energy management systems [50]. Just-in-time heating based on occupant localization has shown up to 7% energy savings [51]. Others investigated the trade-off between achievable energy savings and the risk of comfort impact [52]. Moreover, authors in [53, 54] proposed various solutions of optimizing the compromise between user comfort and energy consumption by taking into consideration occupant interests as well as physical energy and power limitations.

Nevertheless, there is an abundance of literature work that addresses various aspects of intelligent comfort management and energy efficiency maintenance; however, the exhaustive listing of those studies is beyond the scope of this chapter.

2 Smart Building Modeling and Computational System Core

This chapter addresses the core component of any Smart Building (SB), the building management system (BMS). Unlike other designs scrutinized in the literature state of the art, a model-based approach was followed. It facilitates the semantic abstraction of the integrated information and communication technology (ICT) infrastructure as well as the building physical architecture. A high-level functional application programming interface (API) creates such abstraction and enables the development of generic algorithms regardless of the particular SB instance. Such open and flexible BMS architecture is optimized to any building by default, without the need for reengineering and custom deployment. Furthermore, the chapter scrutinizes the design and implementation of an optimized cloud architecture with inherent support for event-driven communication. The provisioned load balancing ensures the scalability regardless of the occupant activities and the complexity of the optimization algorithms. Finally, it concludes with a few case studies, implemented as external API modules, which leverage the BMS-exposed abstractions and provide high-level services to the SB.

2.1 Introduction

This chapter focuses specifically on a critical component of any Smart Building (SB) system, the centralized computational and operation subsystem, the place that coordinates all the building assets and houses any intelligence and optimization logic. For convenience, from now building management system (BMS) term will refer to such component.

The chapter proposes and practically validates an alternative open-source and flexible BMS design named OpenBMS. To the author's opinion, it consists of a highly modular and expandable system without compromising the speed and reliability requirements of any SB management system. The model-based design and the realization of it using a relational database creates a modular system that can be easily extended to model new elements. Additionally, a holistic web application and a real-time application programming interface (API) was designed for facilitating the development of external software logic that leverage the BMS functionality. Finally, care has been taken as well for the creation of a highly computationally efficient implementation with support large buildings, numerous of occupants, and activities even on commodity hardware.

The rest of the chapter is organized as follows. Section 2.2 describes the motivation for pursuing this research direction, while Section 2.3 presents the state of the art in BMS architectures and it compares with the proposed one. Section 2.4 scrutinizes the data and semantic modeling of the SB while Section 2.5 presents in details the system architecture that fulfills the design requirements and implements the previous models. Section 2.6 provides a practical validation of the deployed BMS with two high-level services as case studies that leverage the proposed system and API. This chapter conclusions with Section 2.7.

2.2 Motivation

A major limitation of current BMS solutions is their monolithic, proprietary, and highly unified architectures. On the contrary, the community-driven, open source solutions have not gathered enough momentum outside the hobbyist's cycles yet. On the other hand, the solutions with significant market penetration are either constraint to the software and hardware ecosystem of the manufacturer or hard-configured to that particular building and system revision without any cost effective upgrade possibility.

Model-based BMS design

Thanks to the building data-model formulated during this research, every aspect it, such as occupants, external stakeholders, information and communication technology (ICT) equipment, building structural components and architecture (rooms, floors, walls, windows, doors, etc.) are associated. Therefore, any building with its unique properties and occupants can be characterized by such system-level model that describes the relationships between all

those assets. Such relationships, described in unified modeling language (UML) diagrams can be trivially implemented using a relational database management system (RDBMS). The database implementation is then hosted on the centralized server, which may or may not hold as well the management and optimization logic.

A key advantage of such open, model-based design is its adaptability to any building architecture regardless of the complexity and type of it. For example, the same software can be used without source code modification in industrial, commercial, or residential buildings. Moreover, it can adapt to the size of the building, from a single apartment to a smart neighborhood. Thus, the proposed modeling and design can greatly reduce the cost and complexity and achieve great reutilization of the software modules while designing a BMS system for a building.

Open BMS API

A direct benefit of the API, is the potential for any building stakeholder to interface the assets of the building in a standardized and well-documented way. In plain language, this means that it would not be any more BACnet [55] thermostat, KNX [56] thermostat or even Internet of Things (IoT) thermometer, rather just a kitchen temperature as a service exposed to all management software logic. For example, innovative software that optimizes the energy use while maintaining the user comfort is possible without embedded and network engineering expertise that would have been otherwise required. A well-defined and documented API can eventually lead to an ecosystem of cross-building compatible applications that each of the building stakeholder (inhabitants, designers, energy providers, market regulators) can use for their aims.

High-efficiency BMS implementation

Finally, the RDBMS and the API application servers were designed with performance in mind and optimized for both cloud and localized deployment into embeddable micro-server. Such optimized design is crucial for addressing the adaptability requirements. For example, in the case of a community housing neighborhood of SB that requires remote administration, a powerful cloud server bundled with an embeddable locally installed manager, would better meet the demand of multiple users concurrently requesting for any action or information. On the contrary, an individual apartment, privacy-sensitive occupants, or a corporation could prefer the complete BMS functionality to be hosted on their own protected building premises and communication network.

2.3 State of the Art

Due to the importance of the BMS for the SB, there is an abundance of literature and commercial products on that. This section assesses the comparable solution and highlights the differences with the proposed one.

2.3.1 Scientific literature on BMS

This subsection assesses several research-oriented proposals for BMS architectures that address the different challenges identified by each author. Wang in [57, 58] discusses the challenges of interoperability in a building automation system (BAS). He proposed an alternative to a unifying protocol for the three hierarchical levels of the BAS, two integration layers are proposed instead. At the automation level where BACnet [55] and LonWorks [59] have achieved good interoperability due to their wide adoption, OPC integrates different vendors systems and connects them with the management level. On the latter, the web services are implemented for interfacing the OPC networks.

Jarvinen [60] also addresses the interoperability challenge of a SB. The author proposed ways for interconnecting the non-IP and IP-capable, field level communication protocols through a standard web server interface. On the other hand, Jung [61] used an IPv6 enabled service-oriented architecture to integrate the BAS heterogeneous systems. Finally, Kastner [62] studied the challenges and approaches for seamless integration of legacy BAS and modern IoT. Alternatively, the authors of [63, 64, 65] used ontology-based BMS for semantic abstraction to the heterogeneous network infrastructure.

Many publications also focused on the wireless sensor network (WSN) integration and management. Fortino [66] proposes a framework for managing the WSN to effectively operate a building, while Farias [67] suggested a decentralized approach for control and decision making in smart buildings using WSN. Gisbert [68] focused specifically on industrial applications and proposed an heterogeneous device and network integration platform. Similarly to this work, Stavropoulos [69] presents a system architecture for a university SB. Sensor9k [70] instead is a more complex architecture for heterogeneous WSN and Tragos [71] even presented an IoT-based system, both of which support the development of ambient intelligence applications. Moreover, iPower [72] is another system managing residential energy use that combines WSN and appliance control devices; while Weiss [73] demonstrated a simple web interface and API for controlling power outlets remotely.

Additionally, the energy management and efficiency supporting BMS consists a rather large part of the literature. Hong [74] proposed a cloud-based solution for building energy management for supporting large buildings and numerous occupants. Zhao [75] introduced a conceptual scheme of BMS that focusing on the energy efficiency. Similarly, Gamauf [76] recommended a building load management agent with a generic communication system that enables loose couple between the building and the Smart Grid (SG). Identically, the authors of [77, 78]

designed yet another narrow-scope system for energy management. Chen [79] proposed an architecture that enables the SB infrastructure investments to come in phases while capturing the returns from energy savings and ensuring the occupants' comfort. Finally, Copone [80] presented an architecture that provides a harmonized ecosystem and services for monitoring and control of home appliances energy consumption.

It is obvious that the number of publications on the domain is enormous and quite diversified. Unfortunately, despite the willingness of the author, not all of them can be included in this chapter's state of the art. Nevertheless, during the scientific literature review, some common characteristics shared by most of the publications were recognized.

To begin with, most of them present a BMS of a rather narrow scope and mostly for supporting the specific research objectives (energy management, ambient intelligence, SG interaction, etc.). To the author's knowledge, there is a lack of publication on the system design and implementation of a BMS having conducted beforehand a requirements analysis in a multi-stakeholder environment like the SB, cf. Chapter 1. Furthermore, numerous papers introduced concepts of SB and BMS with only a few practical implementations and validations on a scale. Finally, the papers in the literature can be classified into two major scientific groups based on their focus. The first one concentrates on the legacy BAS and automation science, while the second target the contemporary WSN and IoT to provide the ambient intelligence for the building. However, the author believes that the real value lies in the combination of both and the systemic approach on the SB design [7].

Therefore, this chapter attempts to address the following three limitations. Firstly, it focuses exclusively on the design and implementation of a modern BMS, based on the previous system design and requirements research derived from the SB market and stakeholders analysis. Secondly, the implementation of such design follows the paradigms, recommendations, and lessons learned for large cloud applications and servers for a fast, efficient, scalable, easy-configurable, and cloud deployable architecture. Thirdly, a model-based design and implementation like the proposed is flexible and extensible enough to serve not only new constructions and modern ICT infrastructure but also to retrofit old buildings and integrate legacy automation systems, encompassing and abstracting all those with a common API.

2.3.2 Open-source and community-supported BMS

Unlike the previous subsections that presented mainly scientific work, this subsection collects and assesses the open-source, and community-supported BMS. Those are not always following a structured, scientifically-sound approach, nor they are explicitly focusing on advanced applications such as ambient intelligence or demand response (DR). Nonetheless, they are well documented, easy to get started with, and eventually have attracted the interest of home automation hobbyists and even hardware and software developers that prefer not to create a BMS of their own.

OpenHAB [81] is a BMS software platform which is very popular among the home automation hobbyists. It got traction thanks to the long list of supported devices, interconnecting them under a single platform while also offering different frontend for the machine and human interfacing. For someone with only basic development skills, it is relatively easy to connect devices to the platform and control them remotely. However, it is far from straightforward practice for a typical user, and in the end, it does not facilitate any advanced energy and comfort optimization intelligence. Additionally, while its extendability is exceptional and the compatibility with devices continues to grow, its core architecture does not adequately address the scalability challenges. Nonetheless, openHAB was designed for home automation and hobbyists rather than large SB with complex operations and security requirements.

FHEM [82] is a home automation server written in Perl which allows the configuration of control tasks using home automation devices. It is intended for local deployment in the building servers and supports a couple of building automation protocols. It is still under active development and is a valid alternative to openHAB while having a much smaller community of developers and users.

Domoticz [83] is another popular holistic home automation system for monitoring and control of various devices. Much like the previous two, the community has integrated a number of hardware protocols and developed a simple web interface. Its design objectives, and limitations are very similar to openHAB.

Eclipse SmartHome [84] is open source framework for building home automation systems envisioning to address the highly fragmented market of IoT solutions for the home. OpenHAB is based on Eclipse SmartHome project and its difference with it is that the latter is only a framework to build smart home systems and is not meant to be an end-user solution.

Home Assistant [85] is yet another rising open-source home automation platform for monitoring and control of various devices without the need of cloud deployment. It is written in Python and communicates over WebSockets, a real-time web protocol. It is a very modern, mobile and simplicity-first driven platform that has significant traction in the community as well. Its design objectives are also very similar to the openHAB.

Volttron [86], a distributed control, and sensing software platform with the support of US Department of Energy, on the other hand, focuses mainly on the management of the energy and DR scenarios. It is open, flexible, modular and has various software agents to manage a broad range of systems like the HVAC, electric vehicles and various building loads. It is language and device agnostic enabling the engineers to focus solely on the energy management algorithms and not in interconnection of the various devices. However, this system is mainly tailored to engineers and thus ranks low in the ease of use and configuration by the consumers.

Finally, the BACnet [55] is a more of a traditional building automation standard with an open protocol stack library. It is frequently marketed as the universal building management and automation standard due to its popularity in the domain.

This subsection deliberately does not include the open standards at the device and network layer. Those are not part of the management layer of a SB, even if they can be used as part of the overall SB system. Examples of such standards are the: ZigBee, 6LoWPAN, Z-Wave, LoRaWAN, Sigfox, Thread, EnOcean, AllJoyn, MQTT, Thingsquare, the common Wi-Fi, and possibly many others.

2.3.3 Commercial systems

The last cluster of the BMS solutions are the commercial and proprietary ones that are offered by major consumer electronics manufacturers or standardized in closed alliances. At the time of this writing, they have accumulated considerable interest from the public due to their ecosystem of compatible consumer devices (smartphones, smart entertainment systems, etc.).

Firstly, the Apple HomeKit due to the established ecosystem of smartphone and home entertainment devices is receiving high visibility and popularity. Similarly, the Samsung SmartThings system has its own ecosystem of compatible devices and user base. Both of them are offering a very easy to use and install platform. Despite their ease of use though, they do not provide any clear benefits on the optimized building management aspect as the user has complete control with only simplified energy and comfort optimization algorithms.

Some others notable commercial home and building automation systems revealed during the state of the art study are the following: Belkin WeMo, BuildingOS, Control4, Crestron, Ecobee, HomeSeer, Insteon, Lowe's Iris, Nexia, Nubryte, On-Q, Pella Insynctive, Qivicon, Savant, Skylink, StruxureWare, Switch Automation, Vera, Wink. The list is long and certainly not exhaustive as new technologies and solutions continue to emerge every day.

Finally, proprietary standards and protocols for the device layer of building automation were not scrutinized. Those are standardizing wider or narrower areas of building automation, and they are "open" to various degrees. Some examples for completeness reasons are: KNX, LonWorks, Modbus, M-BUS, DALI, CEBus, C-Bus, SMI, Profibus, AS-Interface, EtherCAT, ControlNet, CANopen, IO-Link, MP-Bus, Interbus.

2.4 Modeling the Smart Building

The modeling of the SB was a critical process during the design of the proposed BMS. As it is already mentioned, the universal data model permits the creation of structured relationships between the physical and digital elements of the building, enhancing the system's adaptability to any type of building and automation ICT infrastructure in place. It enables additionally the storage of the data which characterize a building in a structured manner using a relational database. A significant portion of the developed building model is illustrated in Fig. 2.1 in the form of an UML diagram. The following paragraphs present some key models that form the complete building data model.

Unit and Room models

The core of the building model is built around the notions of `Unit` and `Room` models. Those two have also been the seed and the initial motivation for a flexible alternative to existing BMS. `Unit` denotes any form of building construction with defined stakeholders and management policy. It can be for example the whole building, a building section, an apartment or even a floor. The `Unit` is referenced by the `Room` model which resembles any independently controlled area of the building with a defined interest in the `Unit`'s stakeholders. The `Unit` and `Room` models allow the same BMS to manage, integrate and adapt to heterogeneous living and working quarters and buildings. Building managers can also observe and optimize multiple buildings they are responsible for as `Unit` model is not constrained to collocated spaces or constructions.

Sensor and Actuator models

The second iteration over the realization of the building model involved the ICT devices as they are highly heterogeneous. Each one offers different features, monitored and controlled building factors, and performance. The strategy that was chosen in order to normalize and characterize all those in a relational model was the notion of `Sensor` and `Actuator` models.

The idea behind is the logical separation of any available sensor and actuator interface (e.g., active power, reactive power, temperature, humidity, relay, dimmer, control setpoint, etc.) from the device's algorithm. Each `Sensor` or `Actuator` model not only uniquely characterizes the particular device interface but also permits its direct addressing. However, each model does not hold any protocol algorithm for interfacing the device; for this task a dedicated system has been designed, cf. Chapter 3. On the contrary, each model creates a semantic abstraction of the ICT devices which allows the BMS to interact with them in the form of sensing and actuation services as exposed by those models. Thus, the standardized model's data structures allow the extension of sensor and actuation capabilities independent of the BMS algorithm logic.

Furthermore, such models hold two types of IDs. The first one is the primary key that uniquely identifies the sensing or actuation service-provider to the BMS, while the second acts as the record for the embedded network address (proprietary id, IPv4/6, etc.) of the device. Using the combination of both ID, a commands is routed to the appropriate device. Therefore, a single ICT device can be characterized by a portfolio of uniquely configured `Sensor` and `Actuator` models instances. This composition of model instances is thus characterizing the device as an entity offering the listed services and thus can be adapted to any device. Moreover, if a device with some rare functionality (e.g. radioactivity sensor) not provisioned at this stage is introduced in the future, an update to the `Sensor` or `Actuator` model would be enough for the BMS to support such functionality. Finally, each model references a `Room` one and thus provides to the latter environmental monitoring and control capabilities while localizing the physical ICT instance in the building.

Middleware model

The `Middleware` model has a rather specific scope. It has been developed to support the middleware system that will be scrutinized in Chapter 3. The middleware is a distributed architecture that enables a holistic abstraction of the underlying ICT hardware and building construction to the BMS. Each of the middleware's distributed nodes is assigned to a range or performance limited embedded network. Thus, a `Sensor` or `Actuator` model instance references a `Middleware` one which represents the distributed node. Hence, based on the `Middleware` instance properties, the correct middleware node can be addressed, within which, the embedded network ID stored in `Sensor` or `Actuator` instance will be used to address the correct device.

Therefore, while the middleware enables the seamless connectivity and physical adaptability, cf. Section 3.2, the BMS's model provides the semantic abstraction to those assets and a data model to store, process, and interface the outputs of the middleware nodes and their devices.

Load, Generation, and Storage models

Besides the ICT infrastructure, the building model provides as well data models for the load, generation, and storage physical infrastructure of the building. Those models characterize and store all sorts of information concerning those energy-related entities. The `Load` model enables for example load classification, profiling, and even recognition. IDs are used for uniquely identifying those to the BMS.

The flexibility power of the building model rises from the dynamic association of the `Sensor` or `Actuator` models IDs with the ones of infrastructure models (`Load`,`Generation`,`Storage`). In that sense, the BMS exposes measurement and action services for a given physical infrastructure based on the associated ICT models. Thus, loads can move freely, within and across `Units`, and continue to be monitored and controlled transparently for the BMS intelligence. For example, the process for acting on a load is the following:

- incoming API command on the desired load using its ID or any other uniquely identifying combination of properties that the API supports;
- retrieve the relevant `Load` model instance and the associated and available high level parameters and services (e.g., load controllability, consumption clusters, power, etc.);
- fetch the associated `Sensor` and `Actuator` in order to perform the required action or measurement;
- format the request according to middleware protocol and forward it to the relevant middleware node;
- middleware interfaces the ICT standard and routes the message to the `Sensor`/`Actuator` instance indicated in the model;
- return the result of the operation and through the API, following the reverse process.

This approach of disassociating the physical, energy-related infrastructure from the ICT equipment is an unique advantage compared to state of the art. In fact, it enables even integrated infrastructure controllers to be modeled as a portfolio of `Sensor/Actuator` instances that are associated with a particular `Load`,`Generation`, or `Storage` model. Thus the management algorithms and the API remain the same regardless of infrastructure features.

User model

A holistic building model could not have been complete without the relevant occupant model; the `User` model serves that role. It references the `Unit` model with which the occupant is related with. It holds all the properties relevant to the occupant such as digital access rights (on the server), physical access rights (on premises), location information (in coordinates, or by `Room` reference), per room environmental preferences, per unit management preferences, etc. Furthermore, the `User` model provides the data modeling for storing the outputs of machine learning algorithms and generally the ambient intelligence of the SB. Such data are for example the interests and comfort priorities of the specific occupant.

Building construction models

This corresponds to a cluster of models that characterizing the physical building construction, mainly from the material point of view. Those models enables for example the building-agnostic thermal simulation algorithm. Some models that form this cluster are the: `Zone`, `ZoneBorder`, `WindowAttribute`,`ZoneBorderAttribute`, `MaterialAttribute`. The reader can refer to Section 2.6 for a use case of high-speed BMS-integrated thermal simulation for proactive feedback on occupant energy use.

Virtualization model

Finally, the `VirtualEntity` model enables the real-time virtualization of building ICT equipment, energy infrastructure as well as occupants within an actual physical building and in parallel with its activities. In that sense it creates a "building-in-the-loop" emulation system. The model instances store all the necessary data that parametrize the simulation models. Moreover, depending on the simulated entity, they are referencing the previously presented models which they augment in essence. Thus, the BMS monitors and controls the building energy and environmental assets without knowledge of the virtual nature of some of them, cf. Chapter 4.

Concluding, Fig. 2.1 in the following page and the previous paragraphs are presenting only a portion building model, yet crucial for understanding the potential of the data modeling for a flexible and adaptable SB design. However, those models have not been design in one go. They are the result of incremental updates, extensions, and improvements through continuous feedback from various building stakeholders involved in the design process.

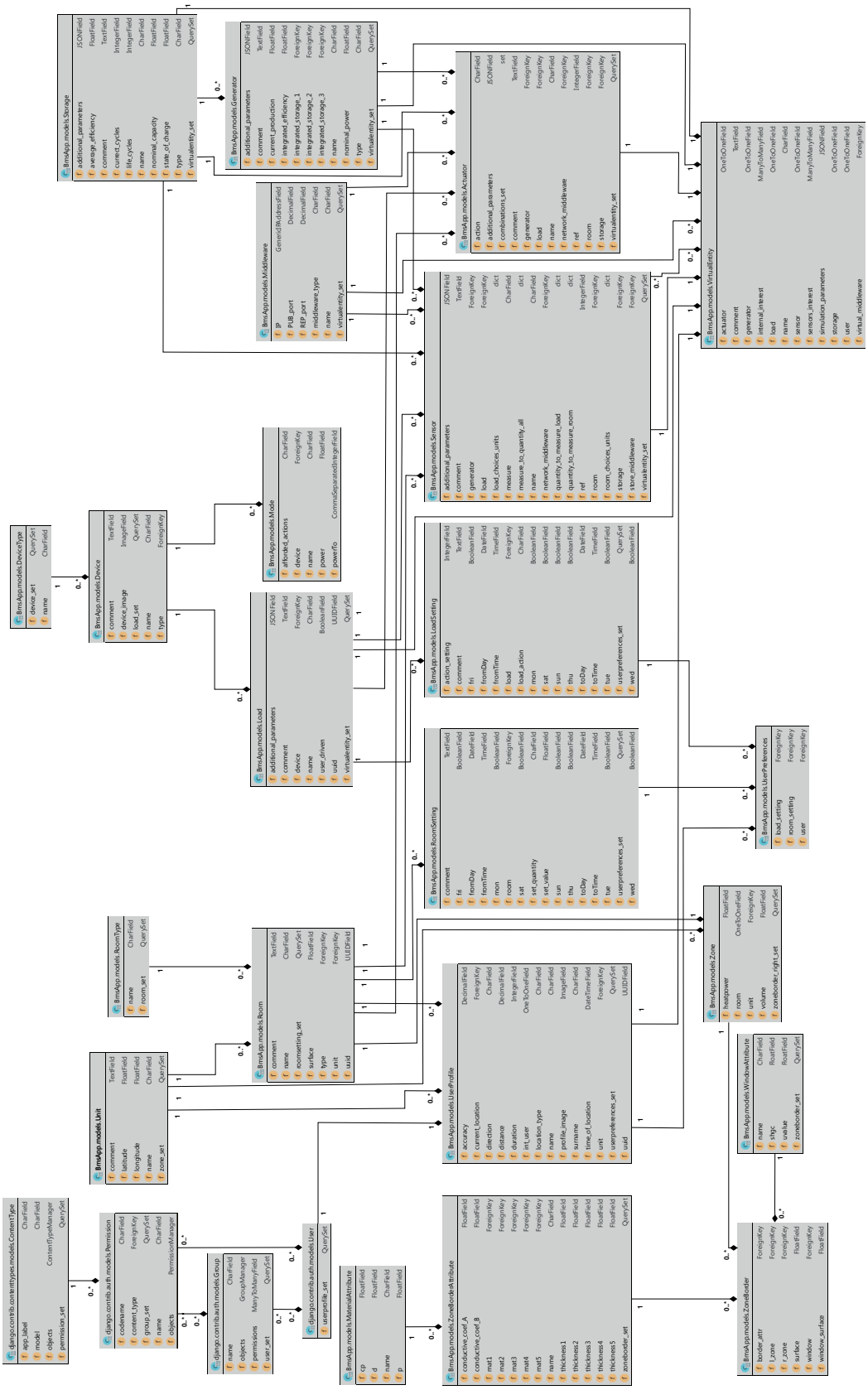


Figure 2.1 – Partial UML diagram of the OpenBMS database model

2.5 Architecture, and Implementation

This sections studies the architecture and implementation of the BMS which is responsible for multiple activities within the proposed SB system such as:

- implementation and operation of the designed building model using a RDBMS;
- design of a web application with a dashboard frontend for SB configuration and demonstration purposes;
- creation and hosting of both a RESTful and a real-time API that exposes the control and measuring services of the SB;
- dedicated, real-time server handling the heterogeneous SB events;
- efficient time series database (TSDB) for recording efficiently the raw real-time data of the ICT devices and occupants. Those data are retrieved by the API for post-processing (data mining, machine learning, energy optimizations).

2.5.1 System architecture, scalability, and rapid deployment

The architecture of the BMS was designed with horizontal and vertical scalability in mind. The system is based on the concept of SB-centered *microservices*, instead of a single monolithic BMS. Hence, specific services or functionalities are scaled up individually during the life-cycle of the building, which facilitates the close match to the BMS computational load. This section presents the overall BMS architecture and highlights how the proposed design meets a growing computational demand by scaling accordingly.

The BMS can be abstracted as a 4-tier architecture as seen in Fig. 2.2. Each tier corresponds to a specific operation of the overall digital system and covers multiple of technologies. Firstly, the *Presentation* tier covers all the technologies for interfacing clients of the BMS either users (human interface device (HID)) or algorithms (API). The *Delivery* tier handles the distribution of the information in the most efficient way possible, ensuring the performance of the system despite of any variation in utilization load and patterns. The *Logic* tier is responsible for the functional management of the SB and hosts the building's optimization software algorithms. Finally, the *Service* tier corresponds to any internal and external services that are crucial for the *Logic* tier technologies.

The overall architecture as a collection of subsystems and *microservices* is illustrated in Fig. 2.3. The primary component in the proposed BMS is the application server for managing the building and providing the synchronous API. A real-time server is also provisioned for the event-driven processes and external applications. Both of them are interfaced with a reverse proxy handling the traffic. Other external databases and the middleware, cf. Chapter 3, are also illustrated. Finally, Docker technology facilitates the rapid deployment, updatability, and management of a large number of BMS servers.

Presentation tier	*Web frontend* *Mobile clients* *IoT* *API*
Delivery tier	*Load balancing across machines* *Caching content* *Ensures performance and scalability*
Logic tier	*Application server* *Real-time server* *Models and data management* *Building management and optimization logic*
Service tier	*RDBMS* *TSDB* *Distributed middleware* *Microservices and third-party services*

Figure 2.2 – The 4-tier BMS architecture

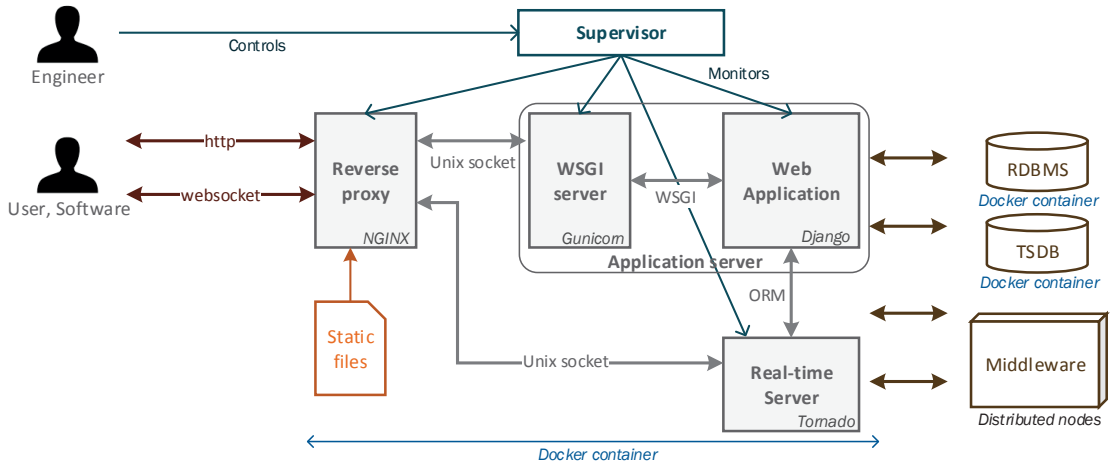


Figure 2.3 – BMS architecture overview

Most BMS in the literature aim to meet the computational demand using a large number of isolated BMS servers to fragment the load, for example per apartment, building, neighborhood. The servers do not share neither the same application logic nor the same data and building states, thus limiting potential optimization benefits from coordinated management of large areas of living spaces.

On the contrary, the author has followed a scalable approach. In that sense, there is a single, yet distributed, BMS server logic to handle the demand. Fig. 2.4 illustrates such scalable design. The horizontal scaling is achieved using a *load balancer* distributing the incoming real-time and synchronous request to the pool of dedicated application and real-time servers. The vertical scaling, on the other hand, is achieved by migrating the Docker containers to more capable hardware in case of a dedicated server, or by allocating more resources in case of a virtual private server (VPS).

The scalability of those computational servers is trivial as they remain stateless. On the other hand, they are sharing the database resources and of course the distributed middleware which represent the stateful side of the BMS. In fact, the middleware, as a cluster of distributed nodes itself, can interface multiple application and real-time servers without an issue due to its queues and asynchronous nature, covered in detail in Chapter 3.

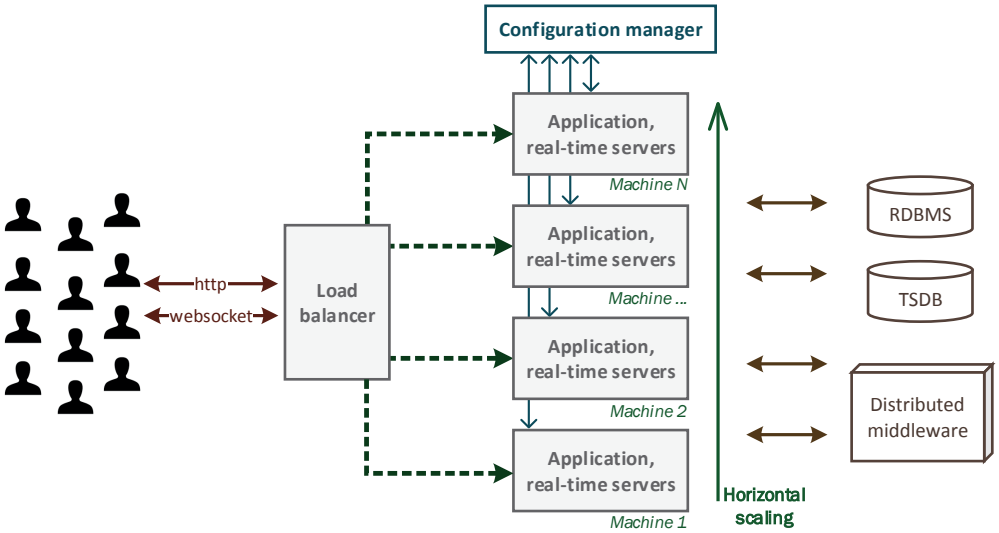


Figure 2.4 – Horizontal stateless scaling of the BMS

The following subsections scrutinize the key subsystems illustrated in Fig. 2.3. Those focus mainly on the *Logic* (application and real-time server) and *Service* (databases) tiers of the 4-tier BMS architecture; the final validation section covers some use case of the *Presentation* tier as well.

2.5.2 Application server as the BMS core

The application server handles all operations between the building resources users (occupants, algorithms) and the backend components (databases, SB middleware, etc.). It hosts the RDBMS for the building models, provides the RESTful API and the minimal building management and data aggregation functionality together with the simple frontend configuration and monitoring dashboard, Fig. 2.5. More sophisticated, management algorithms and frontend interfaces leveraging the exposed SB resources can be implemented using the API.

The web application is based on the well know, feature-rich web framework named Django. It is based on Python and follows the model-view-template architectural pattern. Its rich toolset and the Python programming language has enabled the rapid development of the designed system architecture within a research project time-budget. Moreover, Django is the framework of choice not only for prototyping but also for thousands of production environments in a critical system and web application.

Figure 2.5 – An example frontend as a minimal BMS dashboard

Besides the rapid prototyping and development, Django also has several other advantages that were leveraged for the efficient BMS implementation:

- integrated object-relational mapping (ORM);
- content caching functionality for significant performance gains;
- extensible authentication system;
- built-in template language;
- middleware hooks to alter request or response objects after creation (not to be confused with the proposed SB middleware of Chapter 3).

Architecture

Fig. 2.6 illustrates the core architecture of the application server. The operation is similar to any large scale application server. On an incoming request, the *URL dispatcher*, depending on the URL, invokes the proper *View* function. Depending on the nature of the request, the *View* either renders a frontend template with the dynamic content or JSON-serializes data from the

TSDB or models. The former process is used, for example, when a user requests a frontend view while the latter in the case of an API request. In both cases, the *Caching framework* ensures that identical requests and dynamic data would be fetched from memory rather than rendered or serialized once again.

Concerning the dynamic data, the application server supports multiple sources and external services due to the nature of the BMS. The *Model* corresponds to the semantic abstractions and the ORM through which any data stored in the RDBMS and characterizes the building and its ICT devices is retrieved. Examples of such data as combinations of models are: `Loads` per `Room`, `Sensors` per `Room`, all the temperature `Sensors` in `Unit`, installed `Generation` in `Unit`, all the `Rooms` in the `Unit`, etc. Those can be used either for creating dynamic frontend depending of the installed service in the SB or for replying to specific API requests.

Furthermore, the *TSDatamanager* is responsible for storing and retrieving the real-time sampling data (sensors, actions, occupant locations, etc.) that are placed into the TSDB. It is invoked by the *View* after the *Model* invocation that returns the proper IDs from the RDBMS. Using those IDs the *TSDatamanager* retrieves and stores the data samples on the correct timeseries streams. This step in crucial as the TSDB does not store any relational data or unstructured data that characterize the semantic meaning of the stored streams.

Finally, when it comes to interacting with the physical environment through the ICT devices, or for any other reason requiring to communicate with the distributed middleware, the *View* invokes the *ModeltoBackend*. This process takes place again after the the *Model* invocation in order to retrieve from the RDBMS the internal IDs of the devices, their functionality as well as the middleware node that they belong. The *ModeltoBackend* connects with the middleware using the ZeroMQ protocol and a request(REQ)-reply(REP) socket pair. The algorithmic request of the *View* instance is converted to the proper middleware-compatible message. Last but not least, the*ModeltoBackend* and the *Model* act as the middleware directory service required for the self-discovery functionality discussed in Chapter 3.

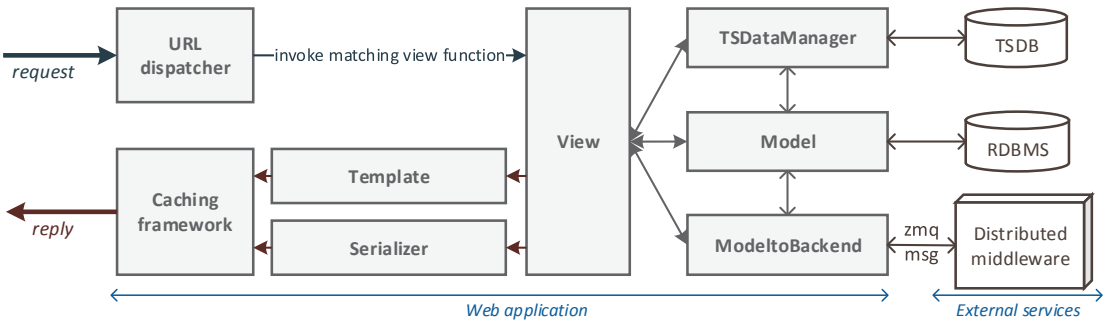


Figure 2.6 – Application server core architecture

Object-relational mapping

The integrated ORM is a considerable advantage of Django framework as indicated before. It automates the conversion of data stored in relational databases into software objects for use in application code. This high-level of abstraction to any supported RDBMS allows the interfacing with scalar data using only Python code instead of merging SQL within. This not only speeds up the development facilitating the management of complex data structures but also permits the switching between various RDBMS without modifying the application code.

Hence, the ORM was used for both implementing and interfacing the data of the SB models studied in Section 2.4. The implementation corresponds to class definitions in Python that follow a relational data model; the committing of the object-oriented models to the relational database tables and relationships is performed with a process called *Migration*. The interfacing of the data corresponds on any use of the ORM in the Python code for combining, filtering, sorting, and any other data operations for retrieving the correct information from the RDBMS.

Django follows a lazy evaluation strategy for improving the database performance. Using the ORM for a database query, the object query is created, passed around, modified, and extended without incurring any database activity. It is only when the query gets evaluated that the final database query is formulated and send to the RDBMS for retrieving the data. Therefore, in the BMS all the database related requests are performed through the ORM which improves the performance overall and mitigates potential poor programming overheads.

API

The holistic API is crucial for exposing to external entities and stakeholders the full length of interaction services of the SB in a trivial and standardize manner. A good API should remain functional, structured, granular yet minimal, and self-explanatory for anyone outside the BMS designers team. Concerning the particular implementation in the proposed system, the API covers all information that can be extracted from the BMS. The implemented list of API resources is reaching 100. All the URLs are well documented with the Swagger library and can be tested directly within a dedicated application web page.

An API can be designed by following various paradigms and implementation schemes. A modern approach for APIs is a REST based API. REST stands for representational state transfer web services and the concept was first introduced in the Ph.D. thesis of Fielding [87]. The REST as a paradigm of web design and not a standard can not be directly compared with established web services standards. Nevertheless, Pautasso et al. [88] illustrated the conceptual and technical differences between the RESTful and the alternative SOAP protocol.

External applications and algorithms have already used the API successfully for many cases such as energy management and optimization, occupant activities and their optimal comfort learning, thermal simulations, mobile applications, and user engaging HID for improved energy awareness.

Security

Django integrates an extensible authentication system. While it is designed for ensuring the access rights to the web application, in the proposed model, the `User` (occupant) is uniquely associated with the internal models of the authentication system. Thus, physical (building) and digital (web application) access rights are unified and co-managed, which enhances the security and transparency. Finally, Django framework has built-in mitigation techniques for cross-site request forgery, cross-site scripting, SQL injection, password cracking and other typical web attacks.

2.5.3 Real-time server

The real-time server enables event-driven processes and external applications to operate over standardized communication patterns without any software engineering tricks. Unlike the synchronous RESTful transactions of the application server, the real-time server interacts using asynchronous messages over the WebSocket standard. This enables an enormous amount of building stakeholders to be updated in real-time with the latest samples from the monitored variables and assets.

Real-time web communication requires long-lived, yet idle, connections to each user of the real-time API. A traditional synchronous server would have required a thread (or micro-thread if supported) to be devoted to each client, an expensive and inefficient practice when thousands of them remain idle.

In order to overcome those challenges, the real-time server integrates a *Tornado* server. The Tornado is a Python framework and asynchronous high performance networking library. Thanks to its non-blocking I/O scheme, it supports tens of thousands of open network connections without latency degradation. The latter makes it an ideal solution for any application requiring long-lived connections such as the real-time server of the BMS. Fig. 2.7 represents the architecture of the real-time server and illustrates the modules that are described in the following paragraphs.

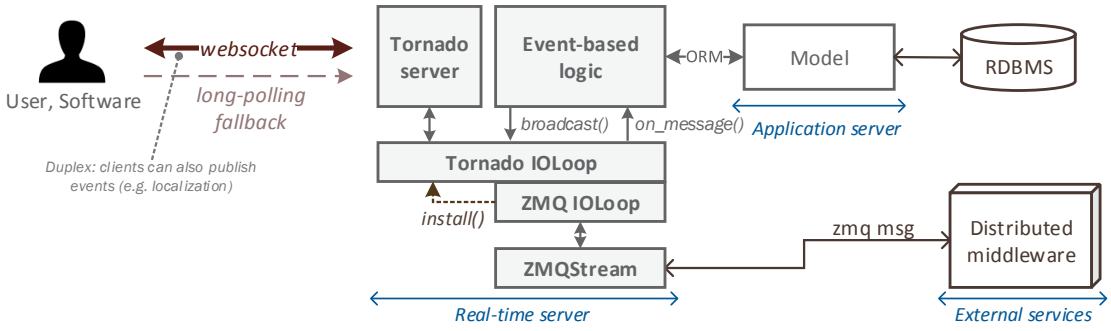


Figure 2.7 – Real-time server core architecture

Firstly, the non-blocking element of the server is called *Tornado IOLoop* and consists of an I/O event loop for non-blocking communication sockets such as the WebSockets. This event loop minimizes the cost of keeping a large number of concurrent connections. Concerning the backend connectivity, the real-time server interfaces the middleware using the ZeroMQ library. It creates a publish(PUB)-subscribe(SUB) socket pair and communicates asynchronously with the middleware nodes. The *ZMQStream* instance registers callbacks for the socket messages' reception. Those callbacks from the ZeroMQ sockets are handled by the *ZMQ IOLoop*. This secondary event loop is intelligently integrated within the main *Tornado IOLoop*. The latter acts now as a universal IOLoop for both ZeroMQ and WebSocket messages, treating any event source with same priority and latency.

The core of the real-time server is pictured in Fig. 2.7 as *Event-based logic*. This module is responsible for processing both sources of events in order to perform the necessary actions. The processing refers to the conversion of the middleware and WebSocket API protocols to the high-level semantic abstractions used by the application server. To achieve that it interfaces the *Model* module of the application server using the common ORM in order to acquire all the model instances necessary to the semantic abstraction. For example, it can convert the message of a power sensor from the middleware and sensor_id combination to the actual load object that consumes it; then it broadcasts that high-level message to all the subscribed API clients. In an alternative example, a WebSocket localization event from a mobile application can be processed, bundled with the matching `User` profile, and then forwarded again to all location events subscribers and external logic for appropriate action (e.g. energy management, comfort, etc.)

Therefore, for the current infrastructure implementation, the real-time server must coexist in the same cloud instance with the application server in order to make use of its ORM for fetching the necessary model instances.

API

From Fig. 2.7 and the previous paragraphs, it is clear that the bidirectional API of the real-time server is over the modern WebSocket protocol, with a fallback unidirectional long-polling support. The real-time API is complimentary to the RESTful one which is primarily used by the client applications when events are not critical to their logic.

More specifically, the WebSocket is a computer communications protocol for bidirectional communication links over a single TCP connection. This TCP-based protocol over common web ports is a benefit for strictly controlled networks. Nevertheless, it is independent of the HTTP protocol which is only used for the initial handshake with the server. Although it is designed for web browsers and web applications, WebSocket can be used by any client software.

2.5.4 Databases

The databases are crucial for both the models and their state storage as well as for keeping track of all the real-time sample data generated within the building. Due to the heterogeneous nature of those two groups of data, a universal database management system is not suitable. Hence, two discrete database management systems have been used and already referred to in the previous sections of the chapter. The following subsections attempt to provide additional background information of those database systems and the rational behind their selection.

Relational database management system

The application server thanks to the Django framework is compatible with a number of different backend RDBMS to better suit individual deployment needs, such as MySQL, PostgreSQL, Oracle, and SQLite. Regardless of choice, the application code does not need modification as the ORM will take care of the actual database queries. It is thus to the discretion of the system engineer to pick the one that better fits the size and topology requirements of the BMS.

For example, the SQLite is a quite powerful implementation of RDBMS ideal for embedded applications and low power platforms. The SQLite database is saved in a single file on a non-volatile memory. The MySQL, on the other hand, is a very popular RDBMS that can be hosted on a server and accessed from multiple web application, hence ideal for load balancing designs. Moreover, the PostgreSQL is an advanced and yet SQL-compliant, object-relational database. Lastly, despite their potential, the author did neither assess nor implement any distributed databases for the purpose of the relational data of the SB. The reason was time constraints as well as the fact that the computational load on this the RDBMS backend, in this application, was never a bottleneck that a vertical scaling could not address.

Time series database

Raw real-time data are always stored in a TSDB. Those NoSQL type of databases outperform the relational ones when they operate on a stream of data advancing over time. Additionally, most TSDB integrate functionality for data aggregation (mean, max, min, etc.), data slicing, and other facilitating the processing of a stream of data monitoring a single variable.

The implementation and characteristics of a TSDB are beyond the scope of this research. Nevertheless, few systems and schemes have been tested for clustering their features and proposing the more suitable ones based on the requirements of the SB.

To begin with, InfluxDB [89] is an open source, self-contained, TSDB. It is fairly simple to setup and is interfaced using its HTTP API. It is also fairly fast and aims to provide the requested data in near real-time manner. InfluxDB is the ideal TSDB for medium to large building sizes and activities. It can be deployed as well in clusters of computing nodes as per load balancing requirements. The OpenTSDB [90] is another large scale TSDB which is written on top of the

Apache HBase, a Hadoop database for distributed, scalable, big data storage. The OpenTSDB is ideal for a massive amount of time series data written even to a millisecond precision. Thus, it is ideal for recording data from clusters of buildings, whole neighbors and even smart cities.

The previous two TSDB are ideal for cloud or centralized management systems. A lighter version of TSDB has been considered for embedded and locally distributed BMS. It is ideal also for short term, cache- and backup-like storage until the data is pushed to the cloud server. The RRDtool (round-robin database tool) [91] facilities the recording of time series data in a circular buffer form. However, it requires the data to arrive a predefined interval or else it interpolates any submitted data. Moreover, it aggregates past data with increasing time step. Thus, in conjunction with the circular buffers for all the aggregation levels, the RRDtool keeps the storage footprint constant over time at the cost of the past data granularity. Lastly, an even simpler approach that was considered is the CSV (comma-separated values) files where each file stores a single parameter and each line of the file is data sample. Apparently, while simple to handle as a storage solution, the overhead of file writing is not ideal for a large number of monitored parameters.

2.6 Functional Validation and Use Cases

This section serves as a functional validation with few BMS-agnostic case studies leveraging the provided API and utilizing the exposed SB assets in order to create more elaborate services. Nevertheless, for a more holistic case study on the whole SB system introduced in this thesis, the reader can refer to Chapter 5.

2.6.1 Simulation-based proactive energy feedback

A significant amount of energy in the buildings can be saved by inducing efficient occupant's behavior [12]. The occupant's awareness tools that have been shown to be effective in achieving energy efficiency gains depend on various computational and estimation algorithms. A model-based, energy feedback scheme was developed for building thermal simulation in order to identify the areas for efficiency improvement. By leveraging the specific mathematical formulation of those models and a dedicated open-source solver, improved computational speed, reduced cost, and enhanced interoperability are obtained. Those combined with the BMS integration through the API, they permit a real-time feedback and facilitate the creation of energy awareness tools with improved accuracy since they rely on validated thermal model simulation and real-time data.

Introduction

There are two strategies for active energy management using the SB infrastructure. While the first leverages the automation to achieve an optimal operation, the second attempts to

bring “user in the loop” for improved energy efficiency, by facilitating user awareness through targeted feedback on his consumption [92]. This study focuses on the development of such a platform.

At the heart of all the energy feedback-based tools lies a computational algorithm, the complexity of which varies considerably depending on the approach, the nature of feedback provided and the available ubiquitous computing at the occupants' living spaces [93]. Recent years have seen the rise of data science-based tools to deliver personalized energy saving reports for the occupants [94, 95, 96]. These techniques mostly deploy disaggregation algorithms on smart metering data to achieve appliance level energy consumption awareness.

However, these techniques necessitate the existence of a usually large input dataset. Moreover, they frequently involve complex calculations in order to extract various patterns from this data, thereby requiring considerable computational power and time, nowhere near the capabilities of currently embedded electronics. Therefore, a fundamental limitation of these solutions is that such information is provided after a significant gap of time and after the energy has been consumed, limiting the potential for action.

This subsection presents an integrated simulation-based platform for providing proactive energy savings recommendations to building occupants with regards to their heating equipment. It leverages the power of ICT enabled sensing technologies, various validated thermal models and an optimized simulation engine built for this purpose.

Background theory

The ability of this integrated tool to deliver accurate feedback rests on the underlying thermal model. It needs to be tailored to the physical properties of the building in question and hence should be able to capture the interactions between physically connected spaces in the building. A typical construction is made up of ceilings, floors, facades, internal walls, as well as windows. All those different elements store heat and transfer it through various mechanisms. Apart from those elements, room air and other masses (ex. furniture) also participate in the thermal process. So a useful representation is to model the heat storage using capacitors and the heat transmission using resistors. This work is built on the well-studied and proved lumped capacitance method. [97, 98]. The choice of this particular model has been motivated by the following considerations:

- The resulting model should be descriptive enough to capture all the relevant dynamics to provide reliable and accurate results. For this, it was necessary to model each room and wall with at least one node.
- It should have reasonable data needs and be computationally efficient to allow for near real-time applications.
- Finally, it should be dynamically customizable for various buildings with minimal overhead.

An equivalent electrical network of resistances and capacitances has been developed to represent the thermal processes in the building. For this, each node is assigned to every room and wall (if the wall has multiple layers then an equal number of nodes can be allocated to the wall), which is then connected to the ground via a capacitor, C.

Heat transfer in a typical building takes place through three processes: conduction, convection, and radiation. Heat conduction across walls under steady state condition can be described by Eq. 2.1. Q_{cond} is the conductive heat transfer rate, k is the thermal conductivity, A and L are the area and the thickness of the wall accordingly, with T_1 & T_2 the temperatures on the two sides of the wall. Convective heat exchange also takes place from the surface of the walls and the room air. This rate of heat transfer is given by Eq. 2.2 where Q_{conv} is the convective heat transfer rate, h is the convective heat transfer coefficient, T_s is the surface temperature and T_{air} is the temperature of the surrounding air.

$$Q_{cond} = \frac{k \cdot A \cdot (T_2 - T_1)}{L} \tag{2.1}$$

$$Q_{conv} = h \cdot A \cdot (T_s - T_{air}) \tag{2.2}$$

In addition to these, heat transfer also takes place via radiation exchange that occurs between the internal surfaces of the wall, the facades surfaces and the sky and irradiation from the sun. The heat exchange between the internal surfaces of the walls is neglected. This is justified since walls of rooms are almost at the same temperature and therefore net heat exchange between them can be neglected. Further, long-wave radiation exchange with the sky can be modeled using a combined convective and radiative heat transfer coefficients for the external surfaces as it has been proposed in [99]. Heat gain from solar radiation can be modeled as direct heat inputs to room air and wall surfaces.

All the above mentioned heat transfer mechanisms, can now be represented using an electrical analogy. In such a model, voltage source plays the role of the temperature of a building element or room, whereas current represents the heat flow. Since resistance is defined as the ratio of potential difference over current, the resistance associated with conduction is given by Eq. 2.3 and the one associated with convection by Eq. 2.4.

$$R_{cond} = \frac{(T_2 - T_1)}{Q_{cond}} = \frac{L}{k \cdot A} \tag{2.3}$$

$$R_{conv} = \frac{(T_s - T_{air})}{Q_{conv}} = \frac{1}{h \cdot A} \tag{2.4}$$

Furthermore, heat storage capacity of walls and rooms can be represented using heat capacitance of capacity Eq. 2.5 where m is the mass and c_p is the specific heat capacity.

$$C = m \cdot c_p \tag{2.5}$$

Furthermore, external and internal heat gains can be trivially added to the model. Internal heat gains and radiators can be modeled as direct power inputs to the room node. This translates into an appropriate current source at the room nodes. For a detailed description for modeling other heat gains, the reader can refer to [100].

It is important to note that in deriving this model, the following assumptions were taken:

- Heat transfer across the walls has been assumed to take place perpendicular to the surface. Thus, there is no variation in temperature over a surface.
- Spatial variations in the temperature of the room have been ignored; therefore, one node is sufficient to represent a complete room.
- The heat capacity of room air has been assumed to be constant at 1.007 $^{kJ}/_{kg \cdot K}$. This is a justified assumption since this value is 1.006 $^{kJ}/_{kg \cdot K}$ and 1.0007 $^{kJ}/_{kg \cdot K}$ at 250 K and 300 K respectively.

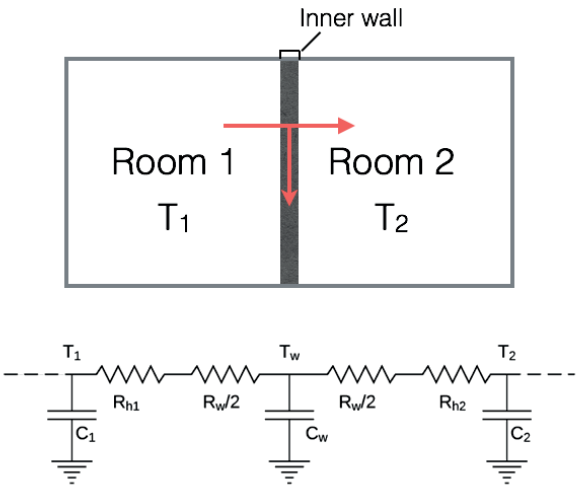


Figure 2.8 – Electrical equivalent circuit to represent thermal processes of an internal wall

Fig. 2.8 demonstrates the modeling procedure using a test case. In this example, there is an internal wall of area A, thickness L and thermal conductivity k. The heat transfer coefficient

on the side of room 1 is h_1 and that on the side of room 2 is h_2. In an equivalent thermal circuit, there are three different nodes with potentials T_1, T_2 and T_w that correspond to the temperatures of air in both the rooms and the wall respectively. Note that the node for the wall temperature has been assigned to the centerline of the walls.

These nodes are connected to the ground via the capacitors Eq. 2.6 and Eq. 2.7. Eq. 2.6 represents the heat storage capacity of air in both rooms and Eq. 2.7 corresponds to the heat storage capacity of the wall. For the equations, ρ_w represents the density and c_w the specific heat capacity of the wall, ρ_a, c_a are the respective ones for the air and $v_{1,2}$ the volume of each room.

$$C_{1,2} = \rho_a \cdot v_{1,2} \cdot c_a \tag{2.6}$$

$$C_w = \rho_w \cdot A \cdot L \cdot c_w \tag{2.7}$$

The heat transfer across the wall has been modeled using the resistances Eq. 2.8 and Eq. 2.9. Eq. 2.8 represents the convective thermal resistance and Eq. 2.9 corresponds to the conductive thermal resistance. This resistance of the wall has been split across the centerline resulting into two thermal resistances of $R_w/2$.

$$R_{h;1,2} = \frac{1}{h_{1,2} \cdot A} \tag{2.8}$$

$$R_w = \frac{L}{k \cdot A} \tag{2.9}$$

A significant advantage of using the above model and approach is that all of the electrical components in the network as well as their parameters have physical interpretations. This direct relation enables the analysis of the effect on any a physical building thermal element by modifying the matching parameter in the electrical network.

Architecture

However, the simulation as mentioned earlier is computationally heavy, and the simulators commonly used in the literature are built for use in specialized mathematical software like Matlab® . As a result, the control and automation modules built around those simulation libraries are also written in the same software language [101]. The shortcoming with the use of such proprietary software packages is that the tools developed in those environments are restricted to just lab level research projects and do not achieve commercial adoption.

This study presents an alternative approach for providing the same thermal simulation outputs regarding the accuracy but with increased performance and usability. The dedicated numerical simulation library of the literature is replaced by a circuit-based simulation engine. Since the problem and model formulation fits excellently with the purpose of the circuit simulator, an increase in simulation performance is expected along with a reduction in overall cost due to the free availability of the circuit solver software.

The integrated tool presented in this case study is composed mainly of the following four discrete agents: the weather related estimation agent, the BMS, the simulation engine and the feedback agent. Fig. 2.9 demonstrates the complete platform architecture.

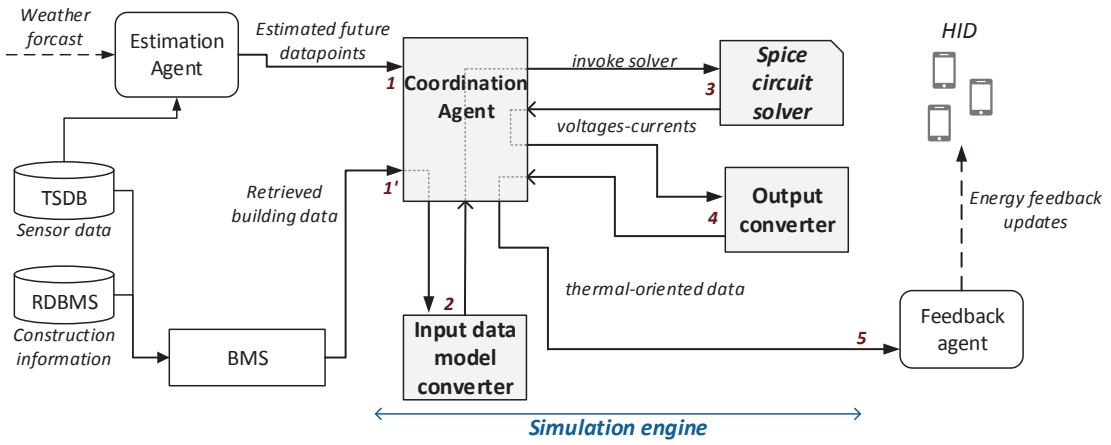


Figure 2.9 – Architecture of the thermal simulation platform

The weather estimator provides the important future environmental data of the neighboring simulated zones for use in the thermal simulation model. To generate the appropriate outputs it takes into consideration the historical values of the zones and the weather prediction from external sources. Using an experimental solar heat gains model, it is possible to estimate within an acceptable error the future temperature data-points of the neighbor zones.

The feedback agent generates intelligent insights and recommendations for the occupant by analyzing the data provided by the simulation engine. As an input, it receives the predicted time series temperature data from the simulation engine and runs the appropriate analysis on them. Its task is to provide the high abstraction level outputs that could be leveraged by

the user devices and in-house displays in order to provide the desired user awareness. The feedback agent is completely decoupled from the inner-workings of the simulation engine. Moreover, it is completely independent from the particular structure of the building and its ICT devices thanks to the BMS. This facilitates the creation or reuse of universal energy awareness applications that have been already proposed or implemented in the literature.

Finally, the core of this work focuses on the design and validation of the simulation engine illustrated in Fig. 2.9. This simulation engine is comprised of the dedicated circuit simulator, a software utility to convert the output of the solver into a usable format, and am engine coordinator agent.

The coordinator agent firstly retrieves data from BMS's RDBMS in order to create the resistance-capacitance thermal model of the building; it automatically synthesizes all its parameters from the building data model. Subsequently, it gathers the environmental data from the weather estimator and TSDB of BMS and creates the dynamically controlled electrical sources of the circuit. The output at this stage is a *netlist* file compatible with the circuit solver (Spice); it describes the circuit to be simulated in full detail. In the next stage, the coordinator agent launches an LTspice [102] console instance and supplies the generated *netlist* file to perform circuit simulation. Once the simulation terminates, the coordinator agent invokes the conversion utility to convert back the circuit related raw outputs to physically meaningful ones. Eventually, the feedback agent pulls these outputs to run its analysis and generate insights for the occupants. The various stages and actions of the coordination are numbered in Fig. 2.9.

Therefore, with the help of the agents developed in this work, the platform can provide proactive energy savings recommendations to the occupants. To use the platform in a new building, the building structure has to be stored in the BMS and the building to be supplied with the necessary ICT devices. Once the system is ready, the occupant only interacts with the platform through the mobile interface by using the feedback agent which returns the recommendations for action. The simulator engine is invoked automatically and the entire process mentioned above is executed.

Results

In order to validate the model, the author performed a comparison of the results of the model with that of state of the art and Matlab-based building resistance-capacitance modeling (BRMC) tool [101]. This tool has been experimentally validated over several months for model predictive control on a real and fully operational office building within the OptiControl-II project [103]. Thus, it provides a sound basis to validate the accuracy of the newly proposed platform.

A test room, Zone 4, has been defined as shown in Fig. 2.10. Two other rooms and a corridor surround this room. Further, for the sake of simplicity floor and ceilings are assumed to have adiabatic boundary conditions. However, they can also be simulated if need be, by two

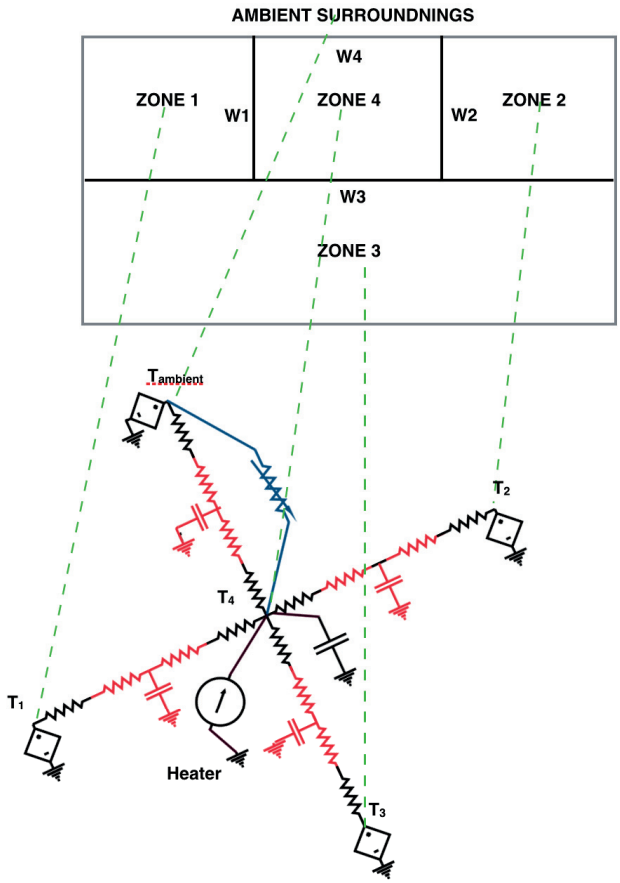


Figure 2.10 – Schematic of a hypothetical room used for validation along with the equivalent R-C network representation

additional circuit branches of capacitances and resistances using the building material and structure specifications of the floor and ceiling.

The temperature profiles of these rooms and corridor along with the outside temperature have been used as boundary conditions for the corresponding walls. This timeseries temperature data was collected from the sensors deployed in one of the EPFL's campus buildings for the month of September 2015. Further, the room is assumed to be unoccupied, with no furniture and no forced ventilation. The material properties used for the simulation are listed in Table 2.1. A window was on wall four which connected the zone under consideration to the external environment. The U-Value of the window was 0.51 $W/K{\cdot}m^2$ with an area of 6.43 m^2. Furthermore, a radiator was present in the room which is modeled as a current source to the room node.

The results of the simulation are shown in the Fig. 2.11 which demonstrates the comparable accuracy of the platform with respect to the state of the art tool. The combination of the free circuit solver with the integrated BMS provides a low cost, yet accurate environment for developing third party applications to realize energy savings through targeted occupant

feedback. These applications can leverage the feedback agent analysis and the near real-time and efficient thermal simulation engine.

Table 2.1 – Building material and structure specifications

	Wall 1	Wall 2	Wall 3	Wall 4
Specific heat capacity $(J/kg{\cdot}K)$	1000	1000	1000	1000
Specific resistivity $(m{\cdot}K/W)$	4.76	4.76	4.76	1.49
Density (kg/m^3)	700	700	700	1600
Thickness (m)	0.1	0.1	0.1	0.27
Area (m^2)	13.40	13.40	9.19	9.19

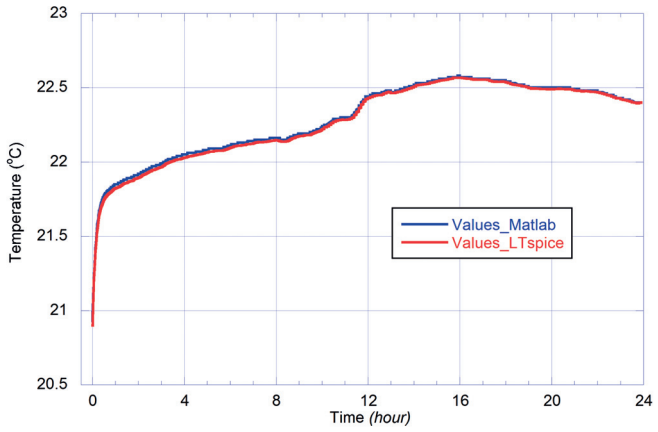


Figure 2.11 – Comparative accuracy of this platform's solver vs a state of the art tool

A specific agent, called "time to temperature" has been implemented. It leverages the ability of this platform and demonstrates its usage. It provides an estimate of the time needed by a specific building zone to achieve the desired temperature. A mobile application is used as a frontend to the user desired temperature set-point. The feedback agent invokes the simulator engine through the REST API and receives the timeseries of the predicted temperature data. It then looks up desired temperature and the delay it took for the building zone to reach it which is displayed to the user. The execution time of the entire process is in the order of 1 sec and hence fast enough for such application.

This particular agent can also be leveraged by third-party applications to generate several insights for the user. For instance, the information of the time it takes to achieve the desired temperature can help occupants save energy since they tend to set higher set-point temperatures believing that it leads to faster heating [51]. This inevitably leads to energy waste if the temperature is not turned down.

Another application envisioned in this work is an energy savings recommendation engine. It invokes the simulator agent multiple times and finds the most optimal control set-points and

actuator positions. It achieves that through sequential thermal simulations for all the different cases and by calculating the energy consumption in each one with the help of a the "time to temperature" agent. The most efficient configuration is recommended to the occupant. Fig. 2.12 visualizes such process in order to calculate the energy saving and best possible scenario for actionable feedback.

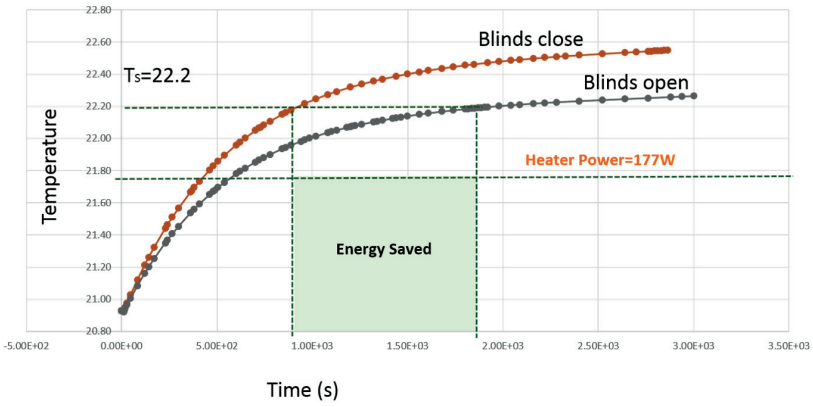


Figure 2.12 – Energy saving recommendations evaluation process

2.6.2 Hybrid, indoors - outdoors occupant localization

Localizing, identifying and authenticating the individual occupants is of paramount importance for the future intelligent buildings. Although the performance of outdoor positioning systems is sufficiently good, the indoor ones have still to converge to a universal technology. This case study proposes a hybrid, unified localization architecture for indoor and outdoor tracking of the building occupants. By taking advantage of the smartphones and their recent near field communication (NFC) capabilities, a low cost, accurate and scalable localization solution is proposed. This system offers location awareness to the presented BMS. It is already deployed in medium scale trial and thus the self-energy use, reliability, ease of use and the privacy requirements are of paramount importance.

Introduction

Despite the advancement in the building intelligence, its performance and optimization potential is bounded by the unpredictability of the occupants. Multiple solutions have been proposed to better understand the residents' usage patterns and reduce the uncertainty. In order to achieve that, it is necessary to know their whereabouts.

However, the available solutions are insufficient, with few if any, holistic and interoperable solutions to accurately monitor occupants' activities. Most of the localizing agents are either custom made, serving a different purpose or are hardly integrated into the existing BMS. In addition, they are exclusively designed either for indoors or outdoors operation. All those

make them highly complex, unable to offer a sustainable and future-proof solution and usually require numerous hours of retrofitting for each BMS.

This subsection presents a unified and efficient localization approach for both indoors and outdoors in order to overcome those challenges and functionally validate the BMS.

Localization methods

The solutions providing localization information can be grouped into four types: the physical, the symbolic, the absolute and the relative location information providers [104]. Regardless of those, they are also characterized based on the particular user and engineering requirements. Fig. 2.13 highlights numerous localization requirements defined by Mautz [105]. Meeting those is a nontrivial task due to the high dimensionality of the technology and its applications. This work's primary aims are the reduced cost, increased integrity, scalability, and privacy. The secondary requirements are met on the best effort basis.

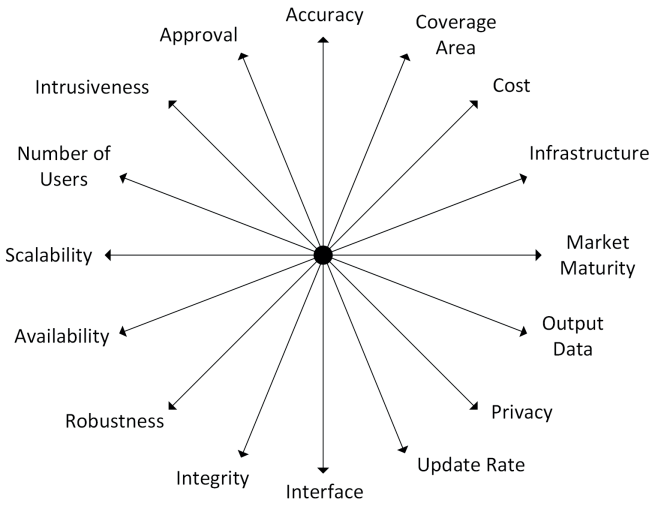


Figure 2.13 – Localization design requirements

The outdoor localization of this module belongs to the physical location type, expressed in geographic coordinates (degree/minutes/seconds). To acquire them, high sensitivity, global navigation satellite system (GNSS) are required. Recently, they have also been enhanced by the Wi-Fi-based positioning systems (WPS) and other positioning identifiers like cell towers IDs and Bluetooth sensors. Those localization systems are mentioned in general under the umbrella term hybrid positioning system (HPS). Well known HPS are the Fused Location Provider by Google® , the gpsOne [106] by Qualcomm® and the Skyhook [107]. The advantages of the HPS are the relatively high accuracy *outdoors* in urban and rural areas alike, rapid localization and good power efficiency, especially compared to GNSS-only solutions.

The literature includes the algorithms for those type of systems [108]. In general, they are based on the multi-lateration principles. One commonly used method is the received signal

strength indicator (RSSI) using the attenuation model in Eq. 2.10 where P_R the received signal strength, d the estimated distance from the emitter, P_T the transmitted power, p the path loss exponent and G_R,G_T the antenna gains of receiver and transmitter respectively. To estimate the position of the receiver, multiple emitters and their respective path loss are considered as seen in Fig. 2.14.

$$P_R \propto P_T \cdot \frac{G_T G_R}{4\pi d^p} \tag{2.10}$$

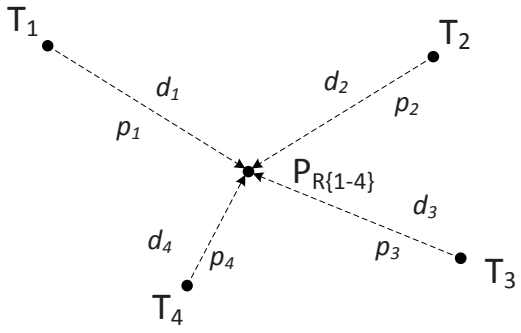


Figure 2.14 – Position estimation using the signal strength information

The HPS has become the ubiquitous positioning method since it is independent of geographical characteristics. However, it does not perform satisfactorily indoors, a tracking principle that was raised in importance the recent years, especially in the domain of SB. Various studies applied the RSSI method in that direction; one has rigorously assessed the localization performance and the causes of its indoors degradation [109]. In summary, the main issues of the RSSI method indoors are the severe path fading, the signal reflections and shadowing. As a result, the perceived reception path losses are very different from the modeled values in Eq. 2.10. Additionally, the variance of parameter p is not only affected by the location in the building but also by the materials involved [109]. In the end, it is not only the accuracy of the system that suffers but also the availability, robustness, and integrity since the GNSS module frequently fails to acquire a satellite lock.

To overcome those limitations, specialized methods and systems have been introduced. They are commonly referred as indoor positioning system (IPS). The IPS can be classified by the medium used for determining the location: infrared, ultrasound, radio frequencies and optical analysis. The literature includes a comprehensive survey on the evaluation criteria for the existing IPS [110]. In particular, unlike the low cost passive radio frequency identification (RFID) for which the tracked object is tagged [111] and the active RFID for which the tracked tag is a transceiver [112], the author proposes a middle-ground solution which merges both technologies and inherits the benefits. The numerous RFID readers functionality is replaced by the NFC capability of the modern smartphones.

Nonetheless, a holistic solution to the localization of the building occupant will require a combination of both indoor and outdoor technologies. This work integrates, enhances them and proposes unified tool as a module of the BMS. An intelligent building must regulate fast, medium and slow relative to time, components. Those demand to change time granularity in tracking and reporting. For example, a heating system would benefit from the long range, outdoor localization and *time to arrive* knowledge, whereas the light control and physical security would benefit from the low latency, indoor tracking. The proposed localization module gradually and transparently shifts from long range low granularity tracking HPS to the IPS for fine indoor tracking even in the most complex buildings. The details of the implementation and design are scrutinized in the following subsection.

Architecture

The `User` model of the BMS includes many configurable parameters and comfort settings, cf. Section 2.4 and Fig. 2.1. The part of it that is relevant to this case study is the universally unique identifier (UUID). This ID is exchanged between the BMS and the localization agent bundled with the location information. Using that the server can recognize and associate the user with his comfort settings for the appropriate actions. Fig. 2.15 demonstrates the initialization and association phase of the mobile application with a `User` model on the BMS side.

The UUID is not a randomly assigned number to each user. Instead, the system uses the UUID stored in one of the user's personal RFID tags. It is up to the user to select one to be identified with. Those can be for example credit card or building access cards that each user possess or will most probably do in the future. For this case study and BMS validation, the university supplied and NFC-capable student cards were used. Due to the sensitive nature of the UUID, all the transactions are conducted through a secured protocol.

Passive RFID tags are placed on or in the walls, on desks or other commonly used places. By the time a user arrives or places his device on one of those, the UUID of both the passive tag and the user stored in the mobile application are transmitted to the BMS server for recognition, authentication, and appropriate action. The sequential diagram describing these client-server transactions and internal operations are visible in Fig. 2.15. This scheme is feasible due to the relational BMS data model that allows the symbolic association of the per-configured, wall embedded, tag location, with the users' UUID leading to a room-level tracking.

A challenge in this design is the choice of the appropriate passive tags that meet the localization design requirements. The NFC is a subset of RFID in the sense that supports mainly the communication standards of ISO 14443 and ISO 15693 on the 13.56 Mhz carrier. On the other hand, the RFID in general supports three different frequency bands, LF (125 - 134 khz), HF (13.56 Mhz) and UHF (856 - 960 Mhz), in addition to various other proprietary protocols. With the cost and the range of communication considered the ISO 15693 standard was chosen. The exact family of integrated circuit (IC) is the Tag-it HF-I Standard by TI® . Fig. 2.16 illustrates the tag with its antenna and dimensions.

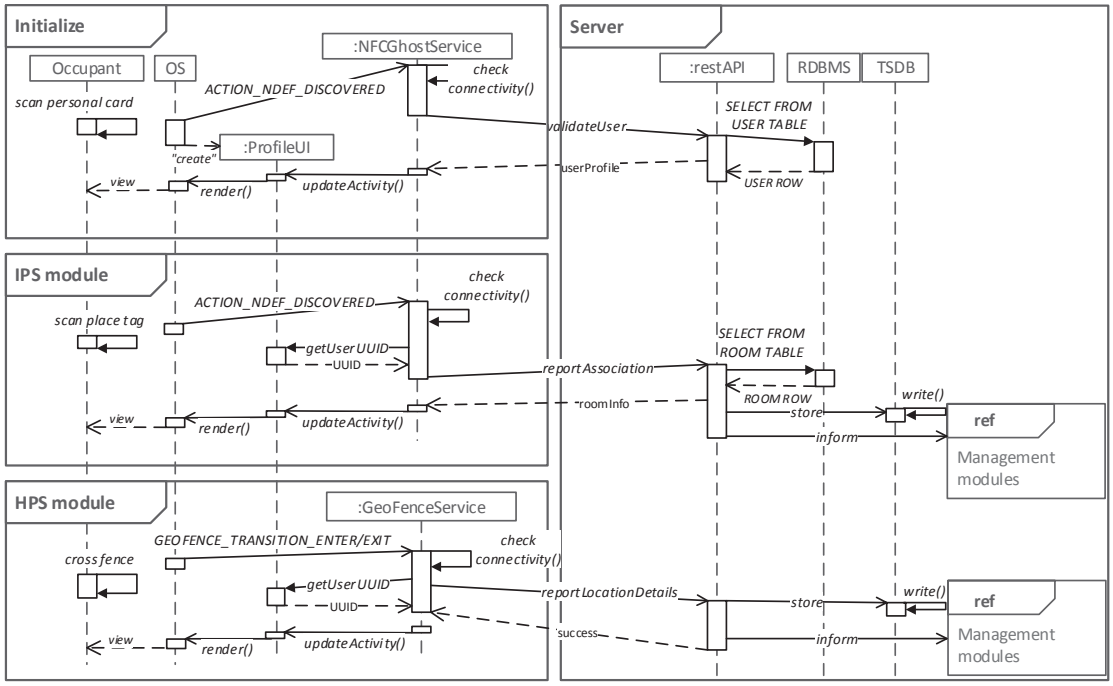


Figure 2.15 – The main UML sequence diagram for (a) User UUID initialization, (b) IPS tag scan, (c) HPS fence event

Figure 2.16 – The indoors localization passive tag

The HPS software agent used for the outdoor tracking is based on the `FusedLocationProviderApi` [113] by Google Location Services. It merges and manages the location providers for each device and provides a simplified abstraction layer to the programming user-space. In that manner, the agent's responsibility is to manage in close collaboration with the BMS the interest points of the user (buildings, workplace, city locations), and to report the location events with low latency.

Furthermore, the HPS agent utilizes a second element called `GeofencingApi` [114]. It permits the creation of radius-based virtual map fences from the previously acquired points of interest. When the device crosses them, even without the application active, the operating system (OS) launches an intent (OS level interrupt) with the current location, accuracy, point of interest that triggered the event, the direction of the fence crossing and expected time to arrive if the direction is inward. The application activates and captures it, informing the BMS promptly. Thanks to this event-driven software design, an accurate outdoor localization and identification can be achieved without battery performance degradation. The relevant

sequence diagram that describes this procedure technically can be seen in Fig. 2.15.

Finally, during the test phase, it was observed that the airplane mode and the restart of the device would erase all fences and nullify any event triggering. Thus, special care was taken to capture the additional events in order to reinitialize the background service and restore all fences. Additionally, in depleted memory situations, the service was developed so that it can freely stop and restarted on demand by the OS memory management unit without any side effects on the localization module state. Finally, one can imagine that the smartphones may experience limited connectivity for example commuting on the subway. Thus the service is designed to locally log the pending events and report them in a batch when it is back online.

Results

To begin with, due to the event-based design principles the IPS module demonstrated *zero* power and computational impact. The application is only invoked when a location tag is present in the NFC field. Additionally, the IPS validation is a trivial task since it is easy to observe the success of the NFC tag reading in software. Experiments using state of the art smartphones demonstrated an average read range of 3cm when the two magnetic coils (tag and reader) were well coupled (parallel magnetic lines) and of 1cm in angled ($\approx 60^{\circ}$) coils. In all situations, the results are acceptable for everyday use with even quick and careless scanning.

To validate the accuracy of the HPS agent in accordance to the BMS requirements, various predefined outdoor paths were followed. Then the reporting time and geographical coordinates were extracted from the database and were appended to the predefined paths and fences radius. An example of the route and location reports is visible in Fig. 2.17. The configuration of the agent is in power balanced mode with *5min* location requests interval. As it is illustrated, by the *green* and *red* markers of entry and exit accordingly, the agent captures the events on time. However, this highly depends on the location request interval, the radius of the inner fence and of course the object speed. Thus, the fences' radius size should be meticulously chosen to improve the localization accuracy.

As already mentioned, the battery impact is of paramount importance as the HPS module is continuously active on the users' device. Therefore, battery usage tests have been conducted for different intervals of location requests. This agent also allows completely passive operation where no explicit location requests are performed. On the contrary, it relies on the location updates requested by other foreground applications, e.g., Maps. As seen in Table 2.2, after a certain threshold in the location requests interval, the energy savings levels off. This is due to the fact that location updates are anyway initialized more frequently by other applications running in the foreground which trigger this module.

Finally, during the validation phase of the system, trial applications leveraging the close integration of the location agent with the BMS were developed. The potential of location awareness in the thermal energy savings has been highlighted before [51, 115]. They

Figure 2.17 – Outdoors localization performance validation

Table 2.2 – Energy impact of various location events intervals

Request Interval	Sleep ratio	Wakeup count	Time spent in 1h
30 s	4.4 %	226	2 min 40 s
1 min	2.1 %	131	1 min 15 s
5 min	0.5 %	57	24 s
15 min	0.6 %	42	20 s
30 min	0.7 %	50	23 s
Passive	0.5 %	54	18 s

demonstrated improvements that did not require any change, neither in the occupant behavior nor in their comfort level. Moreover, they overcame the limits in the per-configured thermal scenarios when the occupant's patterns tend to diverge from the initial ones.

To demonstrate the potential of the localization service for improved energy use, the previous thermal building simulation engine was used. As both are modules of the BMS they are sharing the same data model and API which facilitates their interaction. Therefore, the localization module is tracking the occupant based on the event that are triggered by the HPS module. The later is returning in fact the estimated time to arrive thanks to the inherent support by the Google Location Services. On the other hand, the thermal simulation engine estimates the required time it would need for the space to heat up to the desired temperature stored in the `User` model. By combining both, the BMS is intelligently controlled for just in time heating and thermal comfort, minimizing the pre-heating and adapting automatically to occupants' living patterns.

2.7 Conclusions

This chapter focused on the SB semantic modeling and abstractions through a dedicated flexible data model. Secondly, it scrutinized the required BMS architecture that implements the data model and the core functionality of the SB. The proposed scalable cloud architecture, with inherent support for event-driven communication, maintains its performance and functionality regardless of the occupant activities and the complexity of the optimization algorithms. Up to the time of this writing, the state of the art did not reveal any comparable work, neither in research nor in industrial setting.

The work of this chapter provided an innovative model-based approach on characterizing and interfacing the ICT of the SB. The resulting semantic abstraction architecture is a viable solution for addressing the technology fragmentation of the current market. It is the initial proposal for a system that facilitates the creation of a universal ecosystem of management and optimization algorithms regardless of the particular physical building instance and integrated ICT. In a similar manner to the smartphone platforms, regardless of the mobile hardware, the applications maintain their functionality and portability, increase their public reach, reduce their cost, and guarantee their performance.

3 Distributed Message Oriented Middleware

In the era of the Internet of Things (IoT) and heterogeneous information and communication technology (ICT) systems, monolithic and proprietary Smart Building (SB) systems are unable to address the challenges of extendibility, scalability, adaptability, and security. Improved integration and interoperability of existing and proposed technologies are essential for overcoming the social and financial barriers of SB adoption. This chapter proposes a real-time, brokerless message-oriented middleware (MoM) system for interfacing and interconnecting the digital and physical assets of the SB. It provides a holistic abstraction to the building management system (BMS) of the underlying device protocols and building construction properties, simplifying the design and reducing the overall system cost. Its distributed design adapts and scales to any building construction regardless of the devices performance and connectivity limitations. The expandability is ensured using object-oriented programming paradigms and a layered architecture for each distributed middleware node. A secure architecture ensures the integrity of data and operations, while an extensive performance and energy efficiency study validate the proposed design.

3.1 Introduction

Nowadays, with the technological advancements in ubiquitous computing and automation technologies, Smart Building (SB) has grown beyond wireless sensor networks [116]. The next generation of SB are complex cyber-physical system (CPS) [117, 118]. A unique characteristic of SB's CPS is its extreme variance in topology, scale and involved technologies. This not only complicates the design and development, but it can jeopardize the reliability and efficiency of the management system as well.

Traditionally, embedded systems have always been considered to have better reliability and predictability compared to general-purpose computing [119]. As a matter of fact, the embedded components of CPS introduce requirements largely different from those in general-purpose computing. Therefore, their reliability and predictability should be maintained regardless of the system complexity. Moreover, there are still crucial limitations in their performance, range, and functionality. Another key challenge, especially in Internet of Things (IoT)-based automation systems, is their extreme market fragmentation [120, 7]. There exists a nearly infinite number of competing, non-interoperable solutions without easily identifiable advantages. To make matters worse, those expensive investments become obsolete within a year or two. Additionally, each building's construction is unique and the occupant activities are even more diverse [48, 121, 52]. The topology of the monitoring and control devices [122] as well as the occupant requirements will differ from building to building. However, it is not cost-efficient to design systems, protocols, and solutions with such narrow specifications.

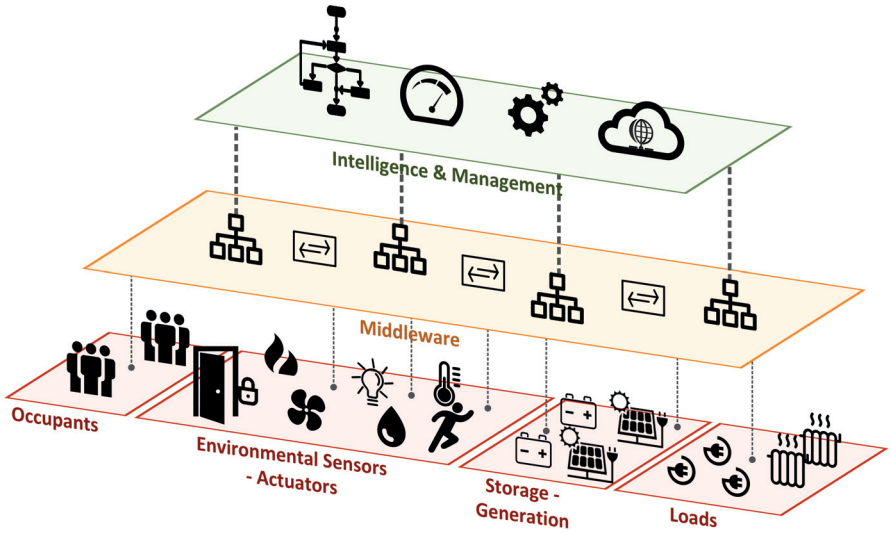


Figure 3.1 – The layered approach in smart building system design

Therefore, any solution effectively addressing the reliability, interoperability, and design adaptability requirements is of high value for sustainable and market competitive building management system (BMS) designs, and above all for the future of SB in general. In fact, Lee [123] scrutinizes the challenges of CPS and how important is the design abstractions.

This chapter proposes a viable solution for addressing those challenges with a middleware communication system. Fig. 3.1 visualizes the concept of middleware within a layered SB design. It provides a universal, flexible and scalable information and communication technology (ICT) abstraction for the high-level entities like the BMS, energy management system (EMS), and any other intelligence algorithms.

The middleware is a well-understood terminology which enables the efficient management of the complexity and heterogeneity of distributed and cloud computing environments. SB systems have similar, yet smaller in scale, challenges and thus the author was intrigued by the possibility of an SB-specialized middleware. Specifically, this work assessed the feasibility of a distributed message-oriented middleware (MoM) architecture. Fig. 3.2 illustrates such distributed middleware deployment across a building floor in order to interconnect incompatible or range and performance limited device networks. Each middleware node, denoted in brown, interfaces one or more device networks for which it implements their protocol stack. They communicate in a peer-to-peer manner over relatively high-performance computer networks, e.g., Wi-Fi, Ethernet, etc. The protocol that governs such communication is universal regardless of the interfaced standard, thus, the middleware enables a protocol-agnostic communication between heterogeneous networks and the BMS. Finally, a low latency and secure distributed communication architecture eliminates the impact of the introduced middleware layer.

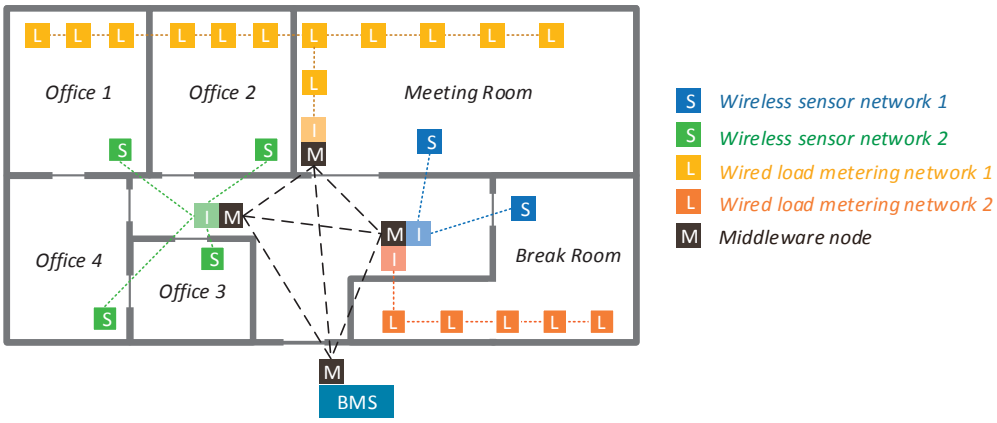


Figure 3.2 – Example of the distributed middleware topology in a building

The rest of this chapter is organized as follows. Section 3.2 analyses the specific requirements of such middleware, which is also the rationale for investing in such technology in SB applications. Section 3.3 assess the state of the art in middleware solutions in general applications as well as specifically for the SB. Section 3.4 dives into the theory behind middleware systems, architecture, and standards. Section 3.5 scrutinizes the proposed middleware design in the scope of SB and analyses the security features of the proposed system. Finally, Section 3.6 presents a detailed validation study on the performance and energy use of the proposed architecture on selected hardware platforms. This chapter finishes with conclusions in Section 3.7.

3.2 Requirements

In general, the target of any middleware is to support large-scale, heterogeneous and distributed architectures. Thus, the requirements for SB middleware reflect the desired functionality of any modern middleware. In this section, these desired features are not only presented but also correlated with SB system requirements in general. By demonstrating in that way the alignment of both, a reader can better understand the motives behind adopting a middleware solution for SB designs.

Interoperability, heterogeneity

The interoperability challenge is well understood for both legacy automation system and newer IoT-based ones. The interoperability with the proposed middleware is achieved by interfacing both specific protocols and abstracting them with a universal data model specifically developed for the needs of SB. The software adaptation layer makes the protocol and data model translation between the two domains. Using this universal data model and internal routing tables, the participating devices are interconnected without any static-configured gateways. Moreover, it enables a complete technology agnostic BMS, as the middleware exposes the monitoring and control capabilities of the building infrastructure. Finally, to further improve interoperability, the middleware is built on open standards which are platform and language independent.

Asynchronous, event-driven communication

All communication is asynchronous with the help of messages and queues, which decouples not only the data model but also the time domain of each interconnected device and network. The latter enables the participation of even the most heterogeneous type of devices. The middleware can provide this event-driven communication without latency or throughput penalty thanks to the high-performance communication layer and libraries.

Mobility, dynamic network topology

The SB is a highly dynamic environment with elevated mobility not only for its occupant but also from the ICT devices if the wearables, entertainment, and intelligent loads are considered as parts of the SB. The middleware enables a dynamic topology where self-discovery, internal addressing protocol, and routing elements allow such functionality. Thus, the continuous operation is guaranteed without BMS reconfiguration.

Scalability, adaptability

A distinct advantage of the particular middleware, as already mentioned, is its distributed nature. Each node of the middleware is distributed in various locations of the building based on the design requirements and the capabilities of the device networks. Those nodes and their interconnection make up the so-called middleware. This distributed design enables a scalable and adaptable solution to any type of building construction, overcoming the embedded network range or design limitations. Additionally, if higher throughput and lower latency are desired from a given embedded network topology, the network can be fragmented into two different middleware nodes for interfacing. Since the middleware nodes communicate over generally superior computer networks, the partitioning of the embedded networks nearly multiplies the overall performance of the initial topology.

Lower cost

The BMS to be installed in residential buildings should remain price competitive. The efficient source code of the middleware and communication libraries guarantees the optimal execution even on the cheapest of embedded hardware. Thus, the introduction of middleware does not increase the cost of the overall system. On the contrary, the interoperability layer seamlessly integrates the existing infrastructure during a retrofitting, reducing the investment size. Moreover, the abstraction layer reduces the development work-hours for the BMS, as it needs to support only the middleware protocol. Finally, the adaptability of the system requires few if any re-engineering to the overall design between different deployments. All those features introduced by the middleware can greatly reduce the overall investment cost and reduce the payback period.

Extendability, ease of development

The SB will continue to evolve over its lifetime and new ICT will eventually be introduced. The term extendability refers to the ease of creating new types of middleware nodes for supporting newer building automation protocols, IoT devices, and wireless networks. More specifically, the software design of the middleware facilitates the extendability of the system using object-oriented programming (OOP) principles and software templates. Therefore, the developer of a middleware protocol node is not required to know how the distributed architectures and messaging communications operate.

Fault tolerance

The middleware nodes can in addition implement localized, narrow scope, control intelligence. In case of a catastrophic failure on the BMS end, the localized management of the premises, for example, the security and safety, will continue to be enforced. In this case, it acts as the backbone of the SB, and the last resort in case of failure. Furthermore, the middleware nodes

can be replicated for redundancy at the middleware level. Lastly, the middleware also mitigates the data loss problem due to interrupted BMS connectivity by locally caching the monitoring data, even without explicit support by the embedded network.

Security

While the security is not exclusive to this middleware, it features some key components towards that direction. Unlike legacy automation systems, the IoT leads the transition from close network topologies to communication over public Internet. The list of potential adversaries ranges from poorly trained personnel or competitors to hackers and other cyber-criminals. The assets and occupants should be protected at all times using validated and standardized technologies and protocols instead of custom proprietary solutions. Thus, the middleware nodes' network is compliant with the security policies in place in order to keep the environment secure at different levels.

Privacy

Ubiquitous computing, despite its advantages, is fairly intrusive to everyday activities. Additionally, the public is fairly privacy-sensitive nowadays. The middleware can implement local data aggregation, anonymization, or even a local management entity without sensitive data leaving the occupants' premises for processing in a centralized BMS server.

Self-discovery, ease of configuration

A key challenge in fully-distributed MoM architectures is the self-discovery of the participating nodes and their services. The absence of a traditional message broker deprives the distributed nodes of a single point of reference. The proposed solution is able to overcome this limitation with the help of the BMS, cf. Chapter 2. The latter serves as a directory and a point of reference for all the middleware nodes. During startup, each node collects the information concerning its peers (e.g. services offered, IP address, ports, etc.). Unlike designs with a dedicated message broker, the directory is involved only during startup. After the self-discovery phase, the nodes continue autonomously and are fully distributed. Hence, this design effectively addresses the discovery challenges of a brokerless MoM while maintaining its benefits, which are scrutinized in Section 3.4.

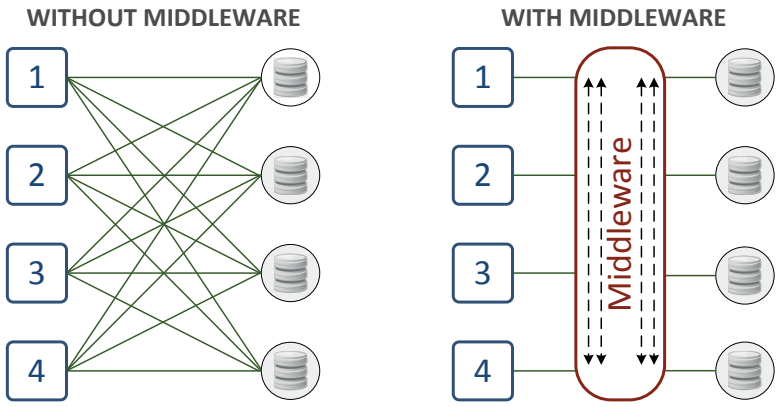


Figure 3.3 – The connectivity advantage of a middleware-enabled system

3.3 State of the Art

To the author's knowledge, the literature does not propose any distributed MoM designed specifically for SB. Although requirements like scalability and interoperability are usually addressed by traditional middleware, reliability and usability, essential for the SB, are largely ignored. This section aims to disseminate the middleware technology landscape and better isolate the distinct advantages of the proposed solution.

3.3.1 Surveys and challenges on middleware design

The design and implementation of a middleware system for CPS is not a trivial effort. Kopetz in his textbook [124] focuses on the design of distributed, real-time embedded systems. It is the tight integration with the physical world that complicates the process. In fact, multiple of researchers have already documented the challenges and approaches for middleware, especially for wireless sensor network (WSN). Hadim [125] investigated for the first time the middleware state of the art, challenges, and approaches for WSN. Hadim also identified the scalability, dynamic network topology, security, and data aggregation as well as the heterogeneity as the key challenges for a successful middleware layer. Moreover, he introduces the concept of virtual database system where the middleware provides an interface for users to extract sensor data. A modified virtual database approach is also shared in this paper using distributed micro-databases for localized data storage service.

Wang et al. [126] proposed a reference framework for analyzing the functionalities of WSN middleware regarding the abstraction and the provided services while classifying the desired features of a WSN middleware. They finally categorize existing work based on this taxonomy and compare their features. Thus, this paper forms a comprehensive state of art and classification for WSN middleware systems.

Ngu et al. [127] in their recent paper focused on the challenges of IoT middleware. They identify the middleware as the key technology of seamless realization of an IoT system tightly

integrated with the physical world. The authors also identify the challenges and features for IoT-oriented middleware such as being a light-weight, application-generic, secure and semantic-enabled design. However, the authors do not focus on SB use of middleware but rather on wearables and health related IoT.

In another recent middleware state of the art review paper [128] the authors studied a very large number of middleware systems, not limited for IoT, on the basis of features such as abstraction, interoperability, context-awareness, adaptivity, service-oriented, computational cost and other. In this paper, the middleware solutions are classified based on the design approaches such as event-, VM- or agent-based, service- or database-oriented, tuple-space, and application-specific. Hence, it provides an interesting overview of the middleware landscape as of 2016.

Chaqfeh [129] for IoT and authors in [130, 131] for WSN studies similarly the challenges and design principles for middleware; in addition, [131, 129] analyze some existing work on the domain. Moreover, Freitas [132] focuses exclusively on the adaptable enhancement of the middleware on WSN, with detailed study and comparisons for a selected number of middleware systems. Finally, Mohamed [133] studies several approaches for service-oriented middleware (SoM) and identifies their requirements and challenges.

Pietzuch in his publication [134] and dissertation [135] presented a scalable, event-driven middleware aiming for distributed computing applications. The research work focuses exclusively on the event-driven middleware design, and thus it is rather holistic in every aspect of a middleware development process. The author studies the requirements for distributed computing, some of which align well, despite the scale difference, with the embedded computing of SB. He presents the concept of overlay networks aiming the abstraction using the logical application-level network on top of the IP topology. Additionally, he introduces the notion of type- and attribute-based publish/subscribe to provide better context awareness and message routing capabilities; those features also adopted to this work. Nevertheless, there are two major differences with the work of Pietzuch; firstly is not addressing the SB or CPS in general, and secondly it is mainly event- and not message-oriented.

Understandably, the number of different middleware solutions, the requirements and their challenges that largely vary depending on the application, requires a considerable effort for analysis. Liu et al. [136] proposed a scenario-based evaluation method for middleware architectures.

3.3.2 Middleware literature for IoT and WSN

The previous subsection documented the review papers on middleware systems, their challenges, and design principles. This subsection lists the primary middleware solutions for IoT and WSN and compares them with the proposed one.

To begin with, Impala [137] is a middleware architecture for application modularity and

adaptation at run time. Applications are transferred to the nodes in native code and linked dynamically during execution in cooperation with other applications. The node can host multiple ad-hoc applications and change between them. The authors of Impala claim easier updates, energy efficiency, and scalability. However, Impala does not support heterogeneity in terms of hardware support.

The MiLAN [138] is another application-oriented middleware, for dynamic management of networks and sensors. It supports service discovery, recognizing newly introduced nodes efficiently. MiLAN architecture expands on the network protocol stack and, like the proposed work, it can operate on top of multiple physical networks. The abstraction is achieved using network specific plug-ins that convert the network packets to MiLAN-compatible messages.

The TinyDB [139] and Cougar [140] are database-inspired approaches to middleware design. Their focus is on architectures for data management for sensor networks using the notion of a distributed database. Both systems support a query language for data streaming. They introduced a distributed query processor at each sensor. However, according to [125], TinyDB is only partially scalable and open to new features since that would require reprogramming of the query processor on all middleware nodes. SINA [141] is another middleware based on a query processing database. Its main feature, over Cougar for example, is the hierarchical clustering of the sensor nodes, aiming for scalability and further energy savings.

The LinkSmart [142] was developed within the Hydra EU project [143] for networked embedded systems. It proposes a middleware that allows developers to integrate heterogeneous physical devices and create ambient intelligence applications using web services for managing the wireless devices.

The SensorWare [144] is another middleware framework for abstracting the sensor runtime, thanks to dynamically defined services. It defines lightweight control scripts for efficient use of WSN computing, communication, and sensing resources. SensorWare is based on the concept of the virtual devices for service abstraction. However, in the author's opinion, SensorWare is not that suitable for resource-constrained hardware.

The Mires [145] is a MoM for sensor networks based on the publish/subscribe communication pattern. It is built on top of TinyOS for addressing the embedded hardware heterogeneity. It is an interesting solution because it is one of the few that has successfully demonstrated a MoM for WSN. It has similarities to this work due to the message-oriented nature. However, it lacks the self-discovery feature, better scalability and most importantly the abstraction capability for more than just embedded sensors.

Finally, the SenseWrap [146], TinySOA [147] and Servilla [148] are three SoM systems for WSN. With those middleware systems, the sensors appear as services to the application level. More specifically, SenseWrap supports service discovery through Zeroconf and uses the notion of virtual sensors for providing communication interfaces over UDP/TCP sockets. TinySOA offers high-level abstraction and service discovery so that applications can access the sensor over an

application programming interface (API). Through that, they can achieve high programming language integration and excellent software interoperability. The disadvantage of TinySOA is the monolithic design that requires not only sensors reprogramming, but also a dedicated gateway, registry, and server. The Servilla is a more recent proposal, the innovation of which is in the concept of in-network service. Instead of hosting the service-oriented logic on external, gateway-like, hardware, each node implements a different part of the middleware and still collectively interact using services.

Finally, it is worth noting that there are some popular message-queue protocols, like the MQTT, which are used mainly for sensors and machine-to-machine (M2M). However, since those are protocols supporting interoperable communication and not middleware for WSN, they are studied in next Section 3.4.

Concluding the state of the art on WSN middleware, it is clear that all of the proposed solutions have a rather narrow application scope. Their focus is solely on addressing the challenges of sensor networks and improving their interoperability. The SB, however, is more than a collection of sensors, actuators, and in general IoT devices. An ideal SB middleware should also consider other potential data sources, e.g. localization data, automation systems, wearables, entertainment and security systems. Additionally, all the above solutions require the middleware as an abstraction layer preprogrammed in the embedded device flash. The proposed solution overcomes this limitation and allows it to be reconfigurable at runtime. Most importantly, this design is not limited to open source WSN like the state of the art, since the core software of the middleware resides in the distributed nodes.

3.3.3 Middleware literature for SB

This subsection collects and analyzes the middleware architecture found in the literature that target a more systemic approach.

Wang [149] follows a web-oriented approach in middleware design. His aim is to integrate heterogeneous, legacy building automation system (BAS) such as the BACnet, LonWorks, etc. However, the solution reads more like a gateway/translator for interconnecting the different protocols rather than a true scalable middleware. Similarly, work in [150, 151, 152] read as in-house interoperability solution rather than a true middleware with many of the identified advantages. Furthermore, LeGuilly et al. [152] used the RESTful architectural principle for providing interoperability. While this is excellent for web services and API implementations, its HTTP base is not suitable for low latency applications.

Patti [153] follows a more interesting systemic approach to middleware for energy efficient buildings. His design and implementation is event-driven, inspired from the LinkSmart middleware described in above. The paper includes also a case study on aggregating data from heterogeneous, software and hardware, sources. While the use of LinkSmart middleware for event-driven communication certainly gives additional value to the paper, the author preferred

to emphasize the building energy-efficiency aspect instead of being an application agnostic middleware for SB.

3.4 Middleware Architecture Standards and Specifications

The middleware concept is well understood, documented and deployed for diverse applications, even from the emergence of the Internet era. Indeed, Banavar [154] praised its integration capabilities for independent applications back in 1999. This section documents and reviews the relevant research work on middleware technologies up to the time of this writing. Those have been the inspiration and scientific guidance for the research and development of the proposed design. A couple of those recent developments have been integrated, and many of the challenges identified in the literature have been addressed.

The following subsections organize the different middleware specifications and literature categorized by middleware type as also documented by [155]. Unlike the state of the art, cf. Section 3.3 which presented final middleware designs or their review papers, this section provides a background overview on middleware architecture standards that govern the development of any middleware system.

3.4.1 Object- and procedure-oriented middleware

The **object- and procedure-oriented** middleware are usually the most mature ones in the literature. They are mostly open source and are suitable for high-performance distributed computing. It is relevant in comparison to the proposed solution due to their performance and excellent programming language support. However, they are less scalable, more complex and tightly coupled, unlike the MoM presented in a following subsection.

To begin with, Common Object Request Broker Architecture (CORBA) [156] is a very popular and mature middleware solution for distributed computing. It is platform and language independent with an object-oriented architecture and excellent programming language integration. The Object Management Group (OMG) has released the specifications as an open standard for accelerating the adoption and interoperability between vendors. Being a mature and established standard has inspired many of the recent work and CORBA-compliant software libraries. However, the standard is inherently complex and extensive. Additionally, according to [135], many-to-many communication pattern is not supported by its broker and has to be simulated by less efficient object services. On the contrary, while this work is featuring neither the same level of popularity nor equivalent programming language integration, it is a much more computationally efficient using messages. This enables the utilization of embedded system for its execution, reducing both cost and energy consumption.

Internet Communications Engine (Ice) [157] is an open source object-oriented middleware. It is influenced by CORBA with which it shares a similar concept. It improves upon the CORBA

object model and provides new features like the asynchronous method dispatch, built-in security and other. It is a rather comprehensive and mature with many programming languages support. However, while it is a high-performance middleware, it is a type of remote procedure call (RPC) framework, and thus it does not meet the low complexity requirement.

Java Remote Method Invocation (RMI) specification [158] is a type of object-oriented RPC framework that synchronously invokes methods of remote objects in different Java Virtual Machines (JVMs) using request/reply communication. Unlike CORBA that focuses on heterogeneous, multi-language deployments and features language-neutral objects; Java RMI assumes the homogeneous environments of JVMs and Java object models. While this enables language-specific optimizations, the developers are constrained to a single programming language.

Microsoft's Distributed Component Object Model (DCOM) is a proprietary technology for distributed objects. It builds on an earlier Component Object Model (COM) architecture for application interoperability in Windows OS environments. The COM models the objects of components, and it uses the distributed computing environment and RPC, together with additional security features, to create standardized network packets conforming to the DCOM standard. Functionally DCOM and CORBA are similar. However, DCOM is proprietary and unlike CORBA, exclusive in Windows OS environment. Despite having closed specifications, DCOM evolves faster, for example, compared to CORBA, due to lack of time-consuming politics involved in generating the next version of specifications [159].

3.4.2 Service-oriented middleware

The SoM on the other hand is based on the notion of providing services remotely through standardized protocols and data models. This type of middleware is a more modern one. Papazoglou [160] studies the state of the art and documents the research challenges in that domain. The vision of the SoM is to provide loosely coupled network services to applications and end-users that create flexible and dynamic processes and agile applications.

The service-oriented middleware shares many advantages with the MoM as it can also use messages for providing those services. However MoM is more suitable for SB needs since the BMS acts already as the service provider for the building, and the middleware nodes need to be as efficient as possible. Nevertheless Al-Jarrodi [161] reviews couple of them, and assesses them versus the service-oriented requirements.

Thrift [162] has been initially developed by Facebook, but now it is an open source project in Apache Software Foundation. This SoM provides the desired interoperability and loose coupling between the nodes. According to [163], it has small memory footprint, asynchronous communication, and adequate performance. Service-Oriented Context-Aware Middleware(SOCAM) [164] is another service-oriented middleware that primarily aims the context-awareness.

3.4.3 Message-oriented middleware

The MoM are of particular interest because the proposed design is of that type. A shared attributed of all MoM is their strong decompiling in communication while maintaining the heterogeneous systems abstraction. There are usually three key subsystems in any MoM:

- *Messages*: They are packets of data exchanged between the middleware nodes; they can be notifications, events, requests for more data or even binaries. There is no hard limit on the size of the message unless it is enforced by the middleware. It is the responsibility of the middleware to fragment the message in network packet payloads for network transportation.
- *Message queues*: messages are exchanged between the nodes with the help of intermediate message queues that decouple the data and execution flow. They hold the sequence of messages waiting to be processed and they provide the asynchronous communication between messaging parties.
- *Message broker*: message broker is an intermediary, centralized server existing in most MoM which coordinates the exchange of messages between parties. A message broker itself has queues for receiving messages but in addition it can perform message validation, translation, and routing.

Thus, the MoM enables loosely coupled distributed software by means of asynchronous messages. The loose coupling of communicating parties has several advantages for a publish/subscribe scheme [165]:

- *Synchronization decoupling*: The sender code does not need to block and wait until the remote code returns. It can proceed regardless of the state of the message and the other node.
- *Time decoupling*: Communicating parties do not need to be active at the same time to participate in the message exchange.
- *Logic decoupling*: They do not need to know each other's software methods in order to exchange information.
- *Space decoupling*: They do not even need to know each other; knowledge of the broker location is enough (broker-enabled MoM).

There are several wire-level protocols for facilitating the development of a MoM, the most prominent of which are the AMQP, MQTT, and STOMP. Those are not MoM systems rather than just messaging protocols. The actual MoM and systems that implement those protocol share many of their features. However, their analysis does not bring any scientific interest in this manuscript. A widespread fault encountered in the literature was the mixing of standards from heterogeneous domains. It is fundamentally different a networking protocol for messages (e.g. WebSockets) with a messaging standard (e.g. AMQP), or a MoM system (e.g. Apache Kafka). The author decided only to scrutinize and compare the major messaging protocols and just

reference the MoM systems that utilize them, skipping the network protocols and language libraries (except ZeroMQ) used for implementing MoM.

AMQP stands for Advanced Message Queuing Protocol, an open specification standard for entities involved in a MoM. AMQP is a binary-based wire protocol designed for high performance messaging and as an interoperable replacement to proprietary messaging standards. It became popular in the corporate world due to its reliability, with hundreds of critical systems relying on it. An AMQP based middleware consists of a broker for routing the messages between the communicating parties and a client library which implements the AMQP protocol. It performs better than other designs with an equivalent feature set, but not as fast as others, as it remains relatively complex and over-sized [166]. RabbitMQ, StormMQ, Apache Qpid, ActiveMQ and Apollo are only some of the messaging libraries that speak the AMQP protocol and support many languages and platforms.

MQ Telemetry Transport, also now known as MQTT, is an open source publish/subscribe protocol specification, originally developed by IBM. What differentiates it from the rest is its design for resource-constrained devices on unreliable, low bandwidth, and high latency networks. Its small footprint on the device cannot be otherwise achieved using the full-featured messaging standards. Its simplicity, low power, and binary packet payload make it an excellent wire-level protocol for integrating the end devices (sensor, actuators, wearables, etc.) with the message broker without additional protocol gateways. RabbitMQ, HiveMQ, Mosquito, Apache ActiveMQ and Apollo are only some of the message brokers that support the popular MQTT protocol.

STOMP stands for Simple/Streaming Text Oriented Message Protocol, a text-based wire protocol. It standardizes the message header and frame body to create a simple and interoperable MoM. STOMP is simple to implement, lightweight and has a wide language support. RabbitMQ, HornetQ, Apache ActiveMQ and Apollo are some messaging systems supporting STOMP.

Finally, the other notable messaging protocols are the Constrained Application Protocol (CoAP), the Extensible Messaging and Presence Protocol (XMPP), and the Web Application Messaging Protocol (WAMP). CoAP is a text-based protocol, similar to HTTP, that is designed for resource-constrained devices. It is best fit for M2M applications like the MQTT, but unlike it, CoAP is primarily a one-to-one protocol. XMPP and WAMP are mainly web-oriented messaging protocols. Despite their popularity to become the IoT protocol of choice, they are not as attractive for large and scalable MoM systems.

Concluding, those standards can be summarized as AMQP and STOMP being suitable for high performance middleware, , MQTT and CoAP being better for embedded devices, and WAMP and XMPP for web services (including resource-capable IoT).

3.4.4 Ideal middleware system and standard

Previous subsections provided a comprehensive overview of the best protocols, standards and middleware systems. However, SB requirements as a CPS and an evolution of traditional BAS do not perfectly align with any of those. The most important difference in SB applications, unlike WSN or distributed computing, is that the system designer is most probably not the developer of the ICT and thus has limited control over their software and data flow. The SB will feature heterogeneous ICT systems that more often than not are of close specification (legacy of the BAS); at best, they may provide a proprietary gateway for external interfacing.

Nevertheless, the author followed a different approach for creating the SB-driven middleware by using the ZeroMQ [167] distributed messaging library and concurrency framework. ZeroMQ is written in C++ and provides the sockets through which messages can be exchanged. It supports multiple transport protocols like in-process, inter-process, TCP, and multicast. The sockets connect in an N-to-N manner with communication patterns such as request(REQ)-reply(REP), publish(PUB)-subscribe(SUB), fan-out, and task distribution. Thanks to the inherent queues, it is asynchronous and scalable to multi-core applications. The documentation, API, and programming language integration are excellent. A distinct feature of the library is its superior message-exchange performance that surpasses many competing and mature middleware protocols, cf. Fig 3.4 [163]. A key characteristic of ZeroMQ is the absence of a broker. Its absence enables the very low latency and high bandwidth. Additionally, it is open source and computationally light, making it an ideal candidate not only for distributed computing but also for CPS. Moreover, the source code of ZeroMQ was successfully ported by the author to even low cost, low power MIPS architecture, which enables a definite advantage over other middleware systems. With all those advantages considered, ZeroMQ was a rational choice for a middleware enabling protocol. The architectural details and implementation are scrutinized in Section 3.5.

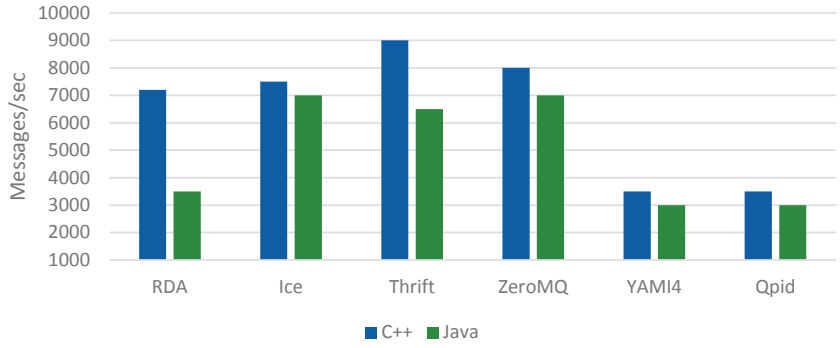


Figure 3.4 – Middleware communication protocols performance comparison

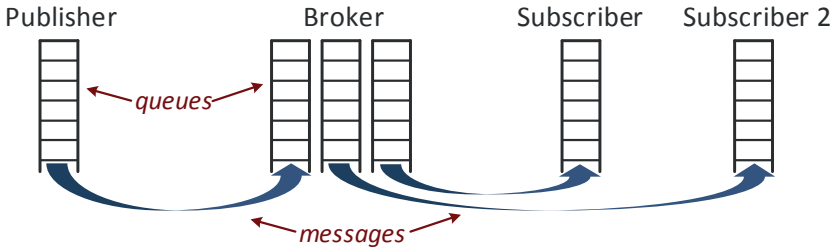
Broker vs brokerless MoM

As a key distinction of ZeroMQ is the brokerless design, it is beneficial for the reader to understand the advantages and disadvantages of this design principle, as well as how the

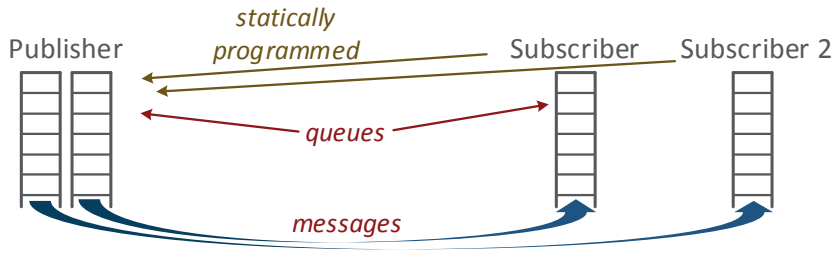
author overcame some of its limitations.

To begin with, a **broker-enabled architecture**, illustrated in Fig. 3.5(a), has several advantages. *Adv1*: Each communicating node does not need to know neither where the others are located, nor how to reach them (space decoupling). The knowledge of the broker is enough. *Adv2*: Enhanced time decoupling. The two nodes do not need to be active at the same time; even their sockets (thus the entire process) may not concurrently exist. The broker can cache and queue the messages. *Adv3*: The broker can do more than message routing and queuing. It can perform protocol translations, message validation and routing as well.

However, it has also some disadvantages for SB applications. *Dis1*: Increased network communication, as messages must first transition through the message broker before they arrive at the designated receiver. This design could be counterproductive if the receiver is on the same layer or shares the same data model (sensor to sensor, sensor to actuator, etc.) as no data frame modification or message routing is necessary. This would be the case, for example, when two similar WSN are interconnected over the middleware, because the embedded network has reached its range or performance limits. *Dis2*: The broker consists of a single point of failure and does not ideally address the requirement for a distributed and possibly localized management of the building.



(a) With broker



(b) Brokerless

Figure 3.5 – Broker-enabled versus brokerless MoM

On the other hand, **brokerless architectures** illustrated in Fig. 3.5(b), are considerably more complex to manage and organize. However, besides the desired full-distributed nature, they have one more key advantage. They are able to maintain very low latencies of communication since there is no intermediary party. This is of paramount importance considering the physical world interaction (CPS) and the various soft real-time control and automation processes taking

place through the middleware.

For this work, a modified brokerless middleware design has been adopted. In order to replace some of the centralized broker functionality, a directory service module has been improvised. Its purpose is to provide a single point of reference for all the middleware nodes as illustrated in Fig. 3.6. The middleware nodes are only pre-configured to search and reach this directory service. This, in turn, has a repository of all the services and nodes of the middleware. Hence, it enables a self-discovery functionality for the proposed middleware. At the same time, the messages continue to be communicated directly between the nodes, retaining their low latency. In the author's opinion, such a design is ideal, as it combines the merits of a brokerless system while addressing the self-discovery challenge of it. The details on the implementation of this directory service are in Section 3.5.

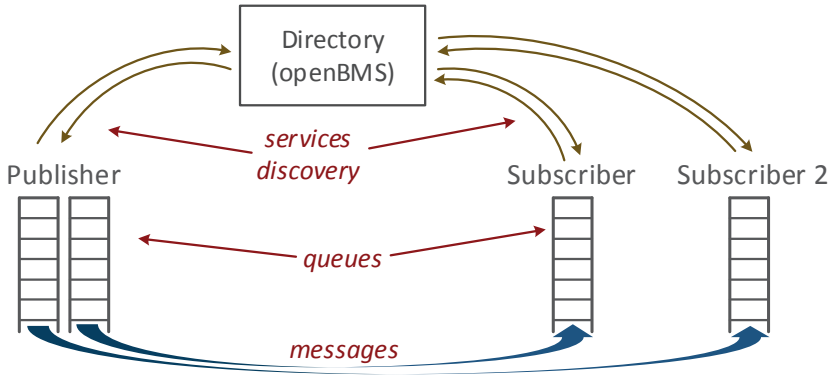


Figure 3.6 – A directory service of the BMS for addressing the disadvantages of brokerless MoM

3.5 Middleware Architecture, Implementation, and Operation

3.5.1 Middleware as part of the BMS

Middleware architectures usually serve the needs of a greater scope system. For the proposed solution, this is the BMS, named openBMS, cf. Chapter 2. A simplified overview is illustrated in Fig. 3.7. The middleware is composed of a number of low power electronics forming the previously mentioned middleware nodes. Those modules are distributed in the building and communicate using the TCP transport and IP network. The data link layers can be either wired 802.3 (100BASE-TX/1000BASE-T Ethernet) or wireless 802.11 (n/ac Wi-Fi). Each middleware node software architecture is analyzed in the following subsection 3.5.2.

3.5.2 Middleware nodes

In order to meet the reduced complexity and easy development requirements identified in the literature as important features for any middleware, a layered software architecture has been followed. The main layers by means of functionality are depicted in Fig. 3.8. The advantage

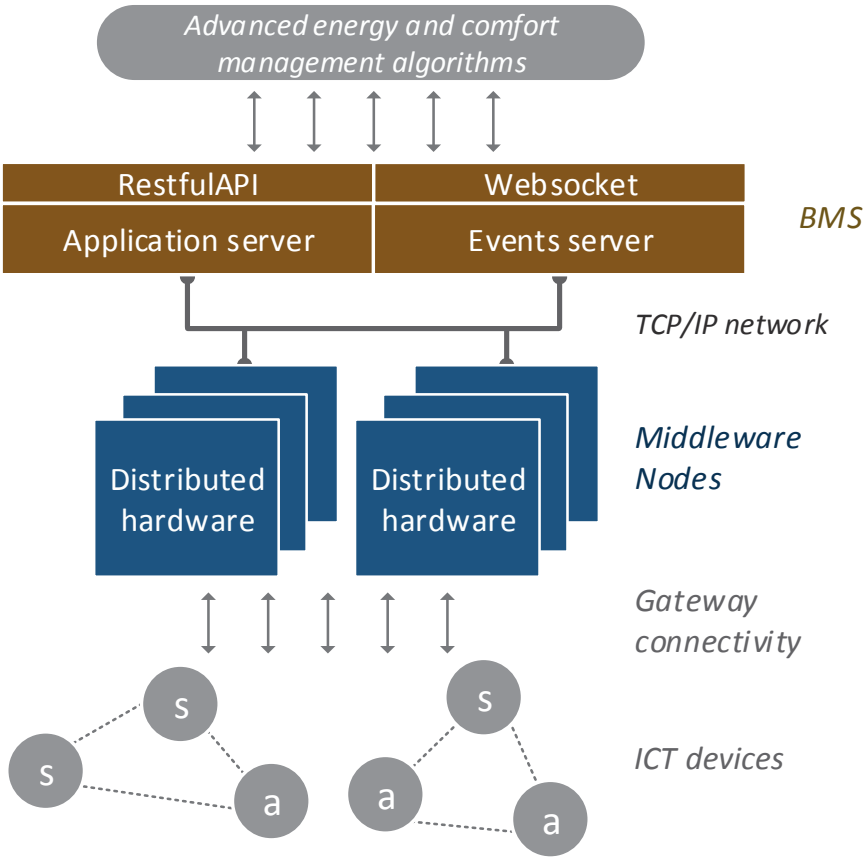


Figure 3.7 – Middleware system in relation with the BMS and embedded devices.

of a layered approach is that for new protocol support, only the 3 bottom layers need to be developed while maintaining the developed top layers related to middleware communication. This enables high reusability of the code, eliminates bug introduction in the middleware connectivity aspect, and minimizes the workhours required for developing a compatibility for a new standard, thus meeting the extendability requirement defined in Section 3.2.

Each node layer performs a specific functionality as described below:

- *Sockets*: the ZeroMQ sockets that are responsible for interfacing with the rest of the middleware nodes as well as the BMS. They are the frontend of the node to the middleware common data space. Each node has 4 of these sockets of different types:
  - *PUB-socket*: a publisher socket is used for low latency events dispatching to any middleware subscriber for such events. Those can be other nodes that engage in automation, a distributed micro-database, or for example, the BMS which needs to collect and analyze any sensing values. The publishing socket *binds* to the [IP:PORT] pair as it is configured in the directory service. Thus, this port should be available on the node and also reachable by all its subscribers.
  - *SUB-socket*: with the subscriber socket, each node can *connect* to one or more

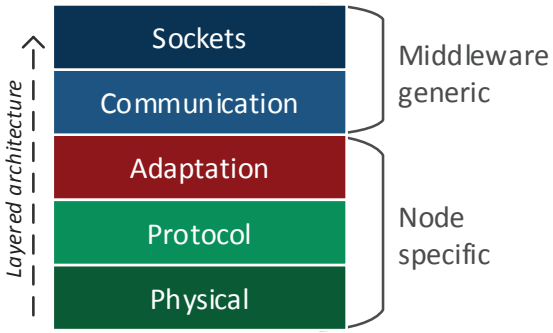


Figure 3.8 – Layered middleware node architecture

publishers to receive events. Data is published along with a topic that lets the subscribers filter the information according to their needs. For this middleware, by definition the subscription topic is the middleware node ID.

  - *REQ-socket*: this is a request type of socket used for *synchronous* communication. It forms a pair with the reply socket type. The pair is in lockstep, meaning that the socket blocks until a reply returns. However, blocking the port does not necessitate a middleware program flow block; a ZeroMQ Poller provides a timeout mechanism for checking if a reply has been received before resuming execution, only to check again in a future time interval.
  - *REP-socket*: the reply socket counterpart for providing requested data. Unlike the publish/subscribe pair, which aims for low latency communication, the request-reply pair is used for reliable bulk data transfers, ensuring the delivery of the payload.

- *Communication layer*: it manages all the communication, synchronization, message encoding, and state maintenance of the node as a participant into the middleware topology. It is an integral part of the middleware node; together with the socket layer they form the middleware data exchange frontend. The software class that implements this layer executes in a dedicated thread in order to respect the timing requirements of the middleware. Additionally, and most importantly, it communicates with the middleware directory service for acquiring the information of the rest of the middleware nodes. The internal operation of this layer, thus the middleware connectivity, is hidden from the other node layers' developers. They just need to use its methods for requesting or submitting high-level data in order to implement the protocol integration or the distributed node-level intelligence.
- *Adaptation layer*: it is the first layer that is responsible for the engineering that extends the middleware functionality or supported standards. The scope of the adaptation layer is to perform the data translation from the node specific to the middleware generic. This layer is necessary for enabling the universal message data standard for heterogeneity between the nodes, regardless of their supported protocol or intelligence. The proposed system provides only the abstract classes that the adaptation engineer should inherit

and implement. The abstraction also standardizes the methods and callbacks that are invoked by the communication layer on incoming middleware messages or by the protocol layer for incoming network data frames. Furthermore, if any form of distributed intelligence is desired, it can be developed on top of the adaptation layer using the middleware generic data protocol.

- *Protocol layer*: it is the equivalent of the communication layer for the physical connection. It implements the complete protocol of the device, network, or database it interfaces. It is independent from the rest of the layers, as its sole responsibility is the state management and coordination of the communication with the physical interface. The design of the protocol layer is left open in order to remain compatible with the maximum number standards, as it is the adaptation layer that defines the data flow between the protocol and the communication layers. Moreover, using multiple protocol layers and physical interfaces, a single hardware can support similar networks and storage backends.
- *Physical layer*: as the name suggests, this is the actual hardware interface to the specific backend entity (device, network, database). It is highly specific to the application and not standardized by any means in the proposed system.

The above text description is summarized and visualized in the unified modeling language (UML) diagram of Fig. 3.9; italic font denotes the abstract classes and methods that need to be implemented. Dark gray depicts an instance of a middleware node which may support one or more protocols or functionality as well as the standardized middleware connectivity. The following paragraphs represent some of the node instances that have been implemented during this work for overall validation of the middleware and BMS integration.

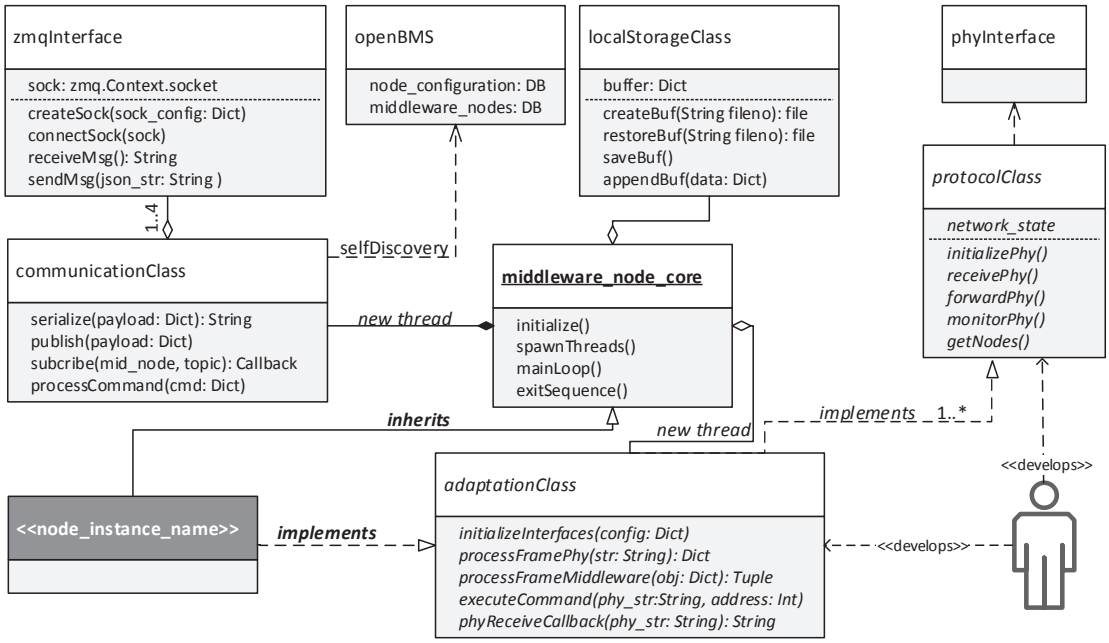
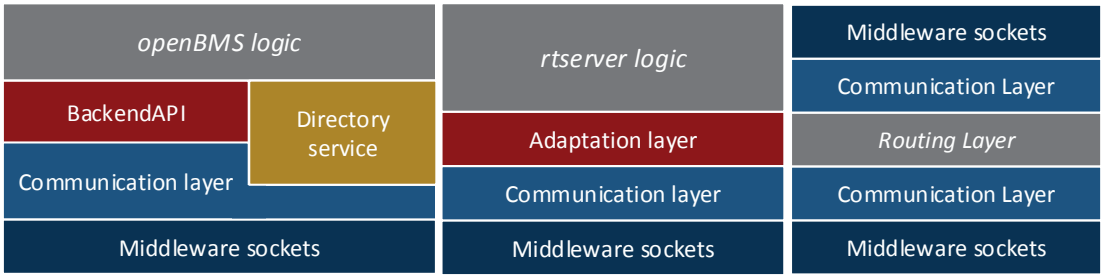


Figure 3.9 – UML class diagram for a middleware node

Firstly, the BMS-side node, cf. Fig. 3.10(a), is the gateway of the BMS server into the middleware topology. The backendAPI layer interprets the server and building-oriented semantics to the equivalent middleware ones. Additionally, it interprets the incoming events and links them to matching building model instances as defined in the BMS. Finally and most importantly, enabled by the openBMS's relational database management system (RDBMS), this node implements the directory service as introduced previously.

The real-time server node, cf. Fig. 3.10(b), is using only a PUB/SUB socket pair. It is responsible for the low latency events and commands coordination across the building. It interconnects and translates the middleware data to the BMS, publishing the events using the Websocket protocol.

The routing node, cf. Fig. 3.10(c), helps to overcome limitation of IP networks like firewalls, NAT, and other network restrictions. More frequently than not, the PUB and REQ nodes port bindings are not accessible from external networks and the cloud hosted BMS. This necessitated the creation of this, externally hosted, routing node. Due to public hosting, its port binding is always accessible. Thus both building located nodes, and BMS can access it. While not always necessary (depends on the system desired architecture), this node acts as a simple message broker for the middleware for the messages needed to traverse a restricted network.

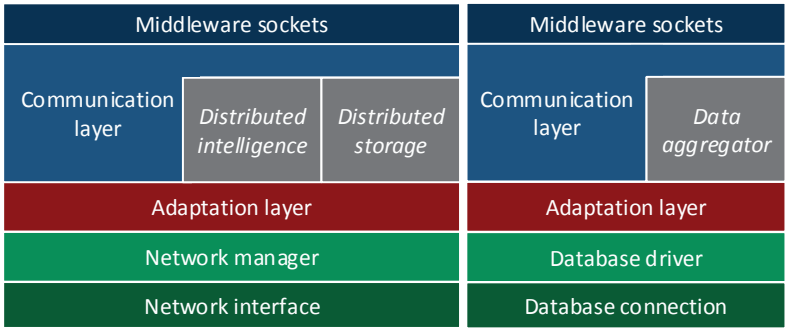


(a) BMS-side node (b) Real-time server node (c) Routing node

Figure 3.10 – BMS-side, real-time server, and routing middleware node architectures

The device/network node, cf. Fig. 3.11(a), is the primary node of the middleware and its inspiration. It handles the data flow and interconnection of the range and energy-limited embedded networks. Moreover, there is a provision for in-network data aggregation, filtering, and minimal preprocessing. At the time of this writing, nodes have already been developed with interfacing capabilities to power line communication, Z-Wave, 6loWPAN and Lora networks.

Finally, the micro-database node, cf. Fig. 3.11(b), is responsible for implementing the on-premises data storage and management. While a cloud hosted, centralized and high performance time series database (TSDB) could perfectly serve the design aims, the distributed one has advantages in data privacy and location. Currently, two data storage solutions have been tested successfully, the CSV and the RRD. Those are lightweight enough even for embedded hardware nodes.



(a) Device/network node (b) Micro-database node

Figure 3.11 – Device/network, and micro-database middleware node architectures

Concluding this subsection, Fig. 3.12 illustrates the sequential UML diagram that describes the network middleware node's main interactions with the BMS and the physical devices. Dark red denotes the startup and self-discovery sequence, a process also shared by the other types of nodes. Black shapes denote the recurring processes of the node and of the BMS.

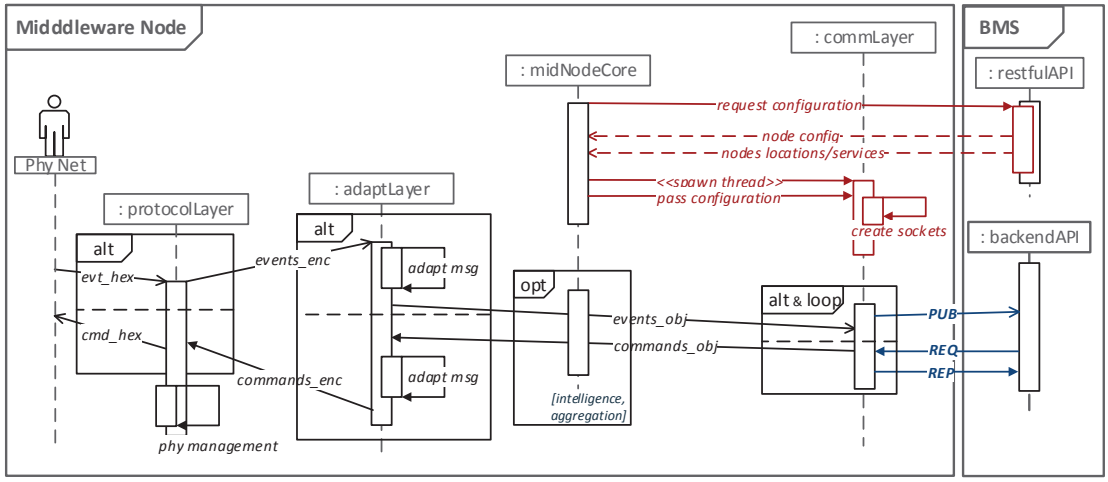


Figure 3.12 – UML sequential diagram of a network middleware node interacting with the BMS and the physical devices.

3.5.3 Self-discovery

The self-discovery feature of the middleware has been mentioned before on multiple occasions. This challenge is encountered as soon as larger distributed architectures are provisioned. Hard-coding the network architecture on each node and manually updating it when a new node is introduced is a valid option. However, this is a fragile and not scalable practice. Imagine a situation with few publishers and hundreds of subscribers. The publishers' IPs are static while the subscribers are dynamic, thus you configure the publishers' addresses for each subscriber. While a new subscriber will share the same configuration, when a new publisher is

introduced, all the subscribers need to get configuration updates. Therefore, the author chose to implement a different approach with the use of a directory service.

The role of the directory as mentioned is handled by openBMS. It hosts in a RDBMS, records for each middleware node. Each record includes not only the IP address but also the four socket information (PUB, SUB, REQ, and REP) as well as the service that it provides for those. Additionally, it stores all the node IDs that the record node needs to subscribe to. Using the node IDs and traversing the DB relationships, the information on the rest of the nodes can be acquired; the complete configuration is then transmitted to the requesting node. For example, a micro-database node fetches not only its configuration for its binding sockets but also the nodes it has to subscribe to in order to collect their sensing data. Concluding the self-discovery, it is by far not the most elaborate self-discovery system. Nevertheless, to the author's knowledge, it is adequate for the complexity and requirements of SB.

3.5.4 Security features

Security of data and infrastructure is of paramount importance for SB with their inherent pervasive nature of CPS. Security is in fact the most challenging of all the requirements; malicious attacks would have a significant impact in an IoT-enabled physical world. Therefore, the system design must not only support the requirements of heterogeneity, scalability, efficiency and anywhere-anytime access, but it is also important to ensure that the security and integrity of the system and data are maintained.

In fact, researchers are very active in the field of IoT security and they have already documented in multiple review studies the security challenges and requirements [27, 28, 26]. Additionally, Roman et al. [25] focused specifically on the distributed IoT which is enhanced with various middleware technologies. However, analyzing specific security frameworks for WSN and IoT is beyond the scope of this work; thus, Sharma et al. [168] work is an excellent overview of the security frameworks for such applications.

Security has a rather large context with, but not limited to, the following characteristics.

- *Authentication*: the process of verifying that someone is who claims he is.
- *Authorization*: the process of ensuring that only the right parties have access to the assets.
- *Confidentiality*: it ensures that the data remain hidden from anyone without the right credentials, using encryption schemes.
- *Protocol and network level security*: describes the mechanics acting within the network to ensure trusted operation while maintaining the power and latency budget.
- *Integrity*: it verifies the reliability of data and refers to the ability to confirm that a message has not been tampered during the transmission.

- *Availability and fault tolerance*: it ensures that the protected assets will continue to offer services even under unfavorable conditions.
- and lastly *privacy*: it refers to a variety of techniques and technologies deployed for protecting sensitive and private data, and communications.

As a matter of fact, the middleware transports the bulk of information in the building and thus can be the weakest point for exploitation by adversaries. Al-Jaroodi et al. [169] review some security middleware techniques and highlighted their characteristics, and challenges.

Some security oriented middleware are the following:

- a security middleware architecture for heterogeneous pervasive devices [170];
- a security management middleware architecture for ubiquitous computing applications [171];
- a middleware for securing the access and control in smart homes [172];
- a secure by design middleware for pervasive computing environments [173];
- and a trust-based middleware for authentication requirements in ubiquitous mobile environments [174];

However, designing a security subsystem from zero requires profound knowledge in the specific domain and years of validation and verification in non-critical systems. The decision has been made to rely on established security standards, architectures and implementations, customizing them for the needs of SB and middleware. The use of validated technologies instead of in-house ones enhances security, reduces risk and promotes an easier security uptake.

The technology that was able to meet most of the security requirements of the specific SB application was the virtual private network (VPN), extended with additional software logic at the middleware node level. The idea behind it is rather simple to understand, yet very effective. Every node connects through an encrypted tunnel to the VPN servers hosted on some designated middleware nodes. The nodes are selected based on their hardware computational power and their location in the distributed topology.

Fig. 3.13 illustrates the scheme and how the VPN is used to support the secure message exchange. The bottom layer is the network topology consisting of hosting hardware, switches, access points, and routes which ensure the physical connectivity of the nodes. The topology is not necessary a local private network since it can extend over public networks (Internet) and to a cloud server if they exist. The second layer, is the virtual network instantiated on top of the underlying network infrastructure. All the message packets are forwarded to the virtual network interfaces of the nodes and exchanged through the VPN servers, also securing the message exchange with the cloud server. The virtual interfaces and thus the nodes behave like

they are on the same network regardless of the public or private underlying networks. The middleware logic operates on top of this virtual network like usual.

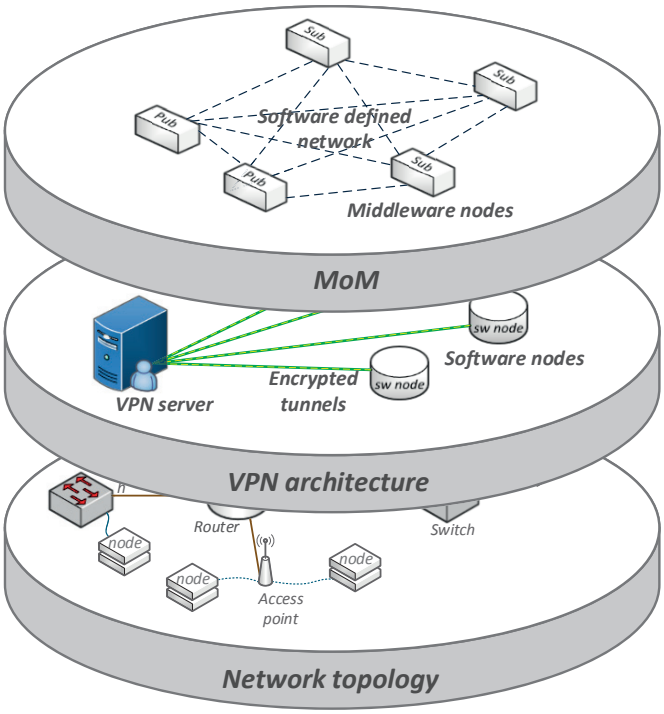


Figure 3.13 – Encapsulation of MoM inside VPN tunnels, on top of existing network

Since all the messages are exchanged over the VPN, it ensures the confidentiality and integrity requirements at all times. The authentication and authorization requirements are ensured by the VPN server accepting the connections. Moreover, using the virtual networks, the designer has the freedom to statically configure the addressing space and packets routing regardless of the network layer, enhancing the mobility of nodes and the networking flexibility of the middleware. The disadvantage is the partial negation of the "no single point of failure" design priority, as messages have to eventually pass through their assigned VPN server node. A failure of such a node would require the served nodes to fall back to their secondary assigned VPN server node, should it exist. Moreover, the VPN overlay may introduce additional delays and requires a capable hardware on the middleware nodes that implement the server.

Regarding the choice of VPN protocol, there are a couple of technologies available which have been considered. Each of them offers a different level of ease of deployment, security, and platform support for addressing different requirements. The article [175] studies the most popular, as of 2004, VPN solutions for Linux systems and compares their network performance.

- *PPTP*: The Point-to-Point Tunnelling Protocol was developed by a vendor consortium founded by Microsoft, Ascend Communications, 3Com, and others. While still popular, it is an obsolete method, published in July 1999 as RFC 2637, for implementing VPN due

to many security issues. The protocol implements only the virtual network and requires additional methods to provide authentication security such as the PAP, CHAP, MS-CHAP v1/v2. The PPTP comes built-in in most operating system (OS)s. Despite the maximum of 128bit encryption keys, security vulnerabilities have been discovered. The most serious, the infamous unencapsulated MS-CHAP v2 authentication exploit required patching through the use of PEAP authentication; Microsoft actively recommends alternative VPN protocols [176]. Moreover, Schneier [177] demonstrated the vulnerability of bit-flipping attack when using the RC4 encryption.

  - Advantages: platform built-in, ease of deployment, the fastest of all protocols [178, 179].
  - Disadvantages: only basic encryption, known vulnerabilities.

- *L2TP/IPSec*: The Layer 2 Tunnelling Protocol, published in 1999 as RFC 2661, does not provide on its own any encryption of the tunneled traffic. It is usually implemented with the Internet Protocol Security (IPSec) for encryption and confidentiality. The L2TP packet is sent withing a UDP datagram, while the negotiation of IPSec security association is carried out over port 500. The lack of support for an alternative port may complicate the deployment behind network address translation (NAT) devices and firewalls. IPSec is still secure without any known vulnerabilities. However, the L2TP provides the tunnel while IPSec ensures a secure channel of communication inside that tunnel, thus L2TP/IPSec encapsulates data twice which may reduce the performance of VPN. Nevertheless, due to the recent instruction set extensions, the hardware can now accelerate the AES encryption implementations. The white paper of Intel® demonstrated a 400% throughput performance gain in IPSec connection in AES-optimized microarchitecture [180]. Finally, L2TP/IPSec does not require any additional software since it comes built-in with many platforms.

  - Advantages: secure, easy to deploy, platform built-in, second faster behind PPTP [181, 178, 179] with multi-threaded kernel support.
  - Disadvantages: double encapsulation, restrictive on usable ports.

- *OpenVPN*: It is an open source technology which uses the OpenSSL encryption library and SSL/TLS for key exchange. It offers possibly the highest security customization with peers authenticating each other using a username/password, a pre-shared secret key, or digital certificates. OpenVPN operates over either UDP or TCP transports and multiplexes the SSL/TLS authentication and key exchange session with the encrypted tunnel data stream. OpenSSL library to provides encryption for both data and control channels and supports several cryptographic algorithms (e.g. AES, Blowfish, 3DES, and others). OpenVPN is fast, but with lower performance compared to the previous two [181, 179] as its current version runs as a single-threaded process. Nevertheless, OpenVPN is highly reliable and stable even on high latency links or over wireless networks. However, while OpenVPN is widely supported, it needs its client to be installed on the system.

  - Advantages: highly configurable, very secure, bypass firewall and NAT device limitations, many encryption ciphers.
  - Disadvantages: requires client installation, more complicated to deploy.

- *SSTP*: The Secure Socket Tunneling Protocol provides means to transport PPP traffic over an SSL/TLS channel similarly with OpenVPN. The use of the standard SSL/TLS TCP port 443 overcomes firewall and other network limitations. Unlike OpenVPN though, it is a proprietary protocol developed and owned by Microsoft® .
  - Advantages: very secure when using strong ciphers, good integration with Windows platform, bypass firewall.
  - Disadvantages: not very suitable for other platforms, proprietary.

- *IKEv2*: Internet Key Exchange (version 2) is a tunneling protocol based on the IPSec. It is a fairly new standard that was co-developed by Microsoft® and Cisco® . Recent versions of Windows support it, and it exists various open source implementations for Linux. IKEv2 has advantages for mobile users in particular due to its capacity to re-establish the VPN tunnel when parties temporarily lose network connection. Security-wise, IPSec is as good, if not superior, the L2TP/IPSec standard. Performance-wise, according to [182], IKEv2 and SSTP demonstrate similar throughput and jitter very close to non-VPN tests.
  - Advantages: faster than other protocols avoiding double encapsulation, very stable with mobile clients, very secure.
  - Disadvantages: not supported in many platforms, server deployment is not trivial.

Considering the above, the decision was to go with the OpenVPN solution primarily for its customization ability and extensive documentation. The somewhat inferior throughput performance [179] is less critical for the mainly low latency but low bandwidth requirements of SB middleware. According to measurements in the next section and in [181, 179] the resulting latency and jitter due to the OpenVPN layer are not significant for building's ICT operations time domain.

Configuring an OpenVPN server for high security is not trivial due to the numerous available parameters and is beyond the scope of this work. Listing 1 displays the exact configuration of the OpenVPN server as a reference to the reader. By default, the OpenVPN server dynamically assigns the client addresses. However, due to the self-discovery requirements, the VPN clients should have a static IP configured according to the directory service. To address these requirements, the parameter `client-config-dir` defines the folder which contains the configuration files matching the nodes' common name (CN). Each file contains a single line: `ifconfig-push 10.8.X.X 10.8.0.1`, where 10.8.X.X is the desired IP of the node and 255.255.0.0 the subnet mask.

Security-wise, the server is configured using a PKI (public key infrastructure) instead of a username/password, defined by the SSL/TLS root certificate (ca), certificate (cert), and private

```
port 1194
proto udp
dev tun
sndbuf 0
rcvbuf 0
topology subnet
server 10.8.0.0 255.255.0.0
client-config-dir /etc/openvpn/ccd
push "redirect-gateway def1 bypass-dhcp"
push "dhcp-option DNS 127.0.1.1"
keepalive 10 120
comp-lzo
persist-key
persist-tun
status openvpn-status.log
verb 3

# Security parameters
ca ca.crt
cert server.crt
key server.key
crl-verify crl.pem
dh dh.pem
cipher AES-256-CBC
tls-auth ta.key 0
```

Listing 1 – OpenVPN server configuration

key (key). While "cert" and "key" are unique to each client and server, they share the "ca". In bidirectional authentication using certificates, the client can also authenticate the server's certificate, much like how a server authenticates the client before mutual trust is established, mitigating man-in-the-middle attacks. An additional advantage of PKI over static keys is that a server can disable access to any compromised middleware node without reissuing new keys for the rest. The `crl-verify` defines such a certificate revocation list (CRL).

Furthermore, for performance reasons communication in the established tunnel is conducted using symmetric encryption. Hence, Diffie-Hellman (dh) parameters are necessary for the symmetric key exchange. Moreover, the ephemeral Diffie-Hellman (DHE) method provides perfect forward secrecy, unlike the plain RSA public key cryptosystem. Perfect forward secrecy is the desired property of a highly secure communication system for which a compromise of a present session key does not compromise past transmitted sessions. The `cipher` defines the desired symmetric cryptography algorithm; OpenVPN supports many of those as seen below.

- DES-CBC, DES-EDE(3)-CBC, DESX-CBC
- AES-(128/192/256)-CBC
- RC2-CBC, RC2-(40/60)-CBC
- BF-CBC
- CAST5-CBC
- SEED-CBC

- CAMELLIA-(128/192/256)-CBC

However, as expected they are neither equally secure nor computationally efficient. In Section 3.6, some of those cipher effects on the middleware performance have been studied. This led to the selection of `AES-256-CBC` as the best combination of performance and security. Finally, the `tls-auth` defines an additional static pre-shared key (PSK) for a HMAC signature to all SSL/TLS handshake packets enabling integrity verification. Any packet without this signature is dropped, mitigating denial-of-service attempts.

Each middleware node gets a VPN client configuration together with its unique private key (key) and certificate (cert) as well as the shared root certificate (ca). The symmetric cipher algorithm should match the server's and thus cannot easily change after deployment.

```
client
dev tun
proto udp
sndbuf 0
rcvbuf 0
remote SERVER_IP_PLACEHOLDER 1194
resolv-retry infinite
nobind
persist-key
persist-tun
remote-cert-tls server
comp-lzo
setenv opt block-outside-dns
key-direction 1
verb 3

# Security parameters
ca ca.crt
cert mid_nodeXYZ.crt
key mid_nodeXYZ.key
cipher AES-256-CBC
tls-auth ta.key 1
```

Listing 2 – OpenVPN client configuration

Concluding, for completeness reasons, an alternative security technology considered was the CurveZMQ. It implements perfect forward security between two ZeroMQ sockets over a TCP connection while maintaining good performance and high security. Curve is a protocol enabling authentication and encryption. It uses short-term session keys for every connection for perfect forward security. The implementation of CurveZMQ also addresses replay, amplification and key theft attacks. However, the VPN option was selected on the basis of its ability to protect against traffic analysis, the ability to secure more than just ZeroMQ traffic, and the author's experience in the field of VPN.

3.6 Validation

The previous sections presented the architecture, implementation, and operation of the proposed middleware design. This section uses that design and deploys it on an embedded hardware for validating the middleware functionality and design requirements.

3.6.1 Evaluated hardware as middleware node platform

Three very different hosting hardware were selected for the purpose providing an holistic testing procedure.

- Intel® Core i5-5300U CPU @2.90Ghz with 16GB DDR3 memory, a x86-64 architecture machine. It used mainly for benchmarking reasons and for setting the absolutely maximum attainable performance record since that capable, yet not very energy and cost efficient hardware, do not meet the requirements of distributed middleware inside buildings. Linux Debian OS was installed at the time of testing.
- BeagleBone Black (BBB), featuring a single-core ARM® Cortex-A8 MPU @1Ghz with 512MB of LPDDR3 memory. This hardware is in the middleground between the high performance computer and the ultra low cost, power and performance embedded electronics. Console-only Linux Debian image was used as the OS during the testing phase. The cost of it is on average $50. The module can be seen in Fig. 3.14, on the left, with attached the USB Wi-Fi interface.
- LinkIt Smart 7688 Duo is a micro-board built around the MediaTek MT7688, a 580 MHz MIPS 24KEc microprocessor unit (MPU), bundled with 128MB memory and a mere 32MB of flash storage. The MIPS system on a chip (SOC) integrates a 1T1R 802.11 b/g/n transceiver and an Ethernet switch. This version is combined on the same board with an ATmega 32U4 microcontroller unit (MCU). The latter greatly enhances the I/O ports and digital communication pins (e.g. I2C, SPI, UART, etc.) while off-loading the main MPU from the real time creating sampling and embedded network management tasks. The MPU and MCU communicate over high-speed UART, thus inter-communication is not an issue. It runs a highly stripped-down GNU/Linux-based firmware that is commonly used in network router hardware called OpenWRT. Moreover, it includes a microSD interface, a handy feature for a micro-database node functionality. At the time of this writing, the cost of this board is $16; a very competitively priced solution considering its capabilities. Fig. 3.14, on the right, pictures the module without the Ethernet breakout board for providing the RJ45 connector.

3.6.2 Performance and validation tests

In order to assess the performance, various benchmarks have been conducted as follows. The tests have been performed for each of the proposed hardware for comparison reasons and for

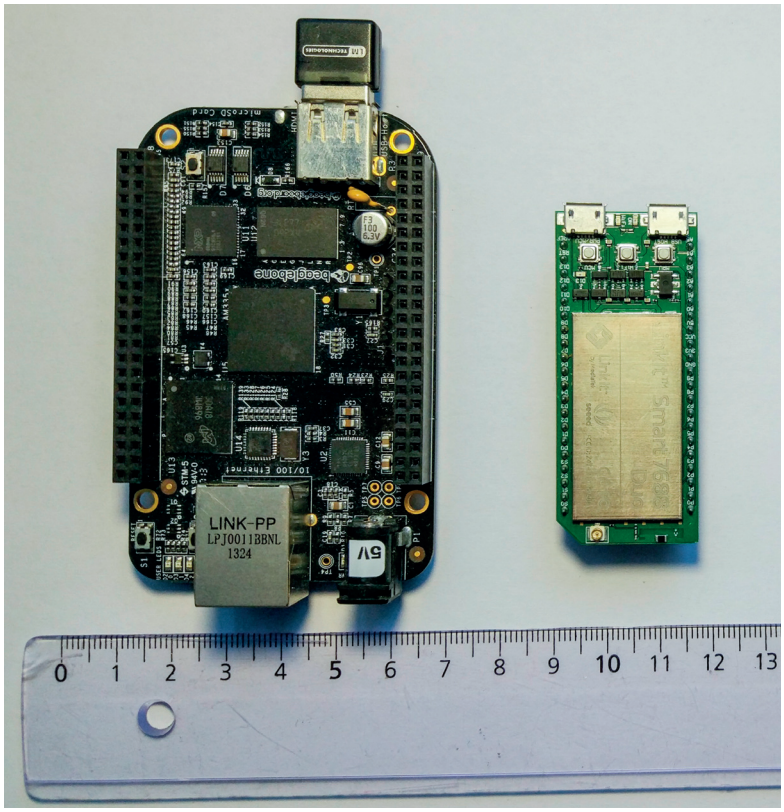


Figure 3.14 – Embedded MPU boards for hosting the middleware node software and the physical interfaces. Left: BeagleBone Black (BBB), right: LinkIt Smart 7688 Duo

validating the presented design.

- Round-trip time (RTT) and maximum TCP throughput assessment of a network link between the embedded hardware and a reference one, an Intel® Core i7-6700 CPU @ 4.00Ghz with 32GB DDR4 memory. This test defines the reference values for best latency and throughput that can be achieved with the given hardware and network. It is essentially crucial for evaluating and normalizing the middleware performance of this configuration.
- Middleware message throughput and latency assessment for unencrypted communication.
- Cryptographic performance evaluation on this resource-constrained node. Those tests are necessary for understanding the performance degradation as a result of the additional VPN layer.
- RTT and maximum TCP throughput assessment of the encrypted tunnel defining the encrypted latency and throughput.
- Middleware message throughputs and latency tests repeated over the encrypted tunnel.

- The section concludes with an energy assessment and the effect of programming language on the performance of the middleware node.

Algorithm 1, developed in both C++ and Python, was used to assess the latency. It is measured using a REQ/REP socket pair by sending and echoing back several messages. The round-trip time average calculates the latency; divided by two, it provides the average latency for a packet transmission.

Algorithm 1 Message latency assessment

1: **procedure** MSG_ECHO(url, msg_{count})
2: $s \leftarrow$CREATE_SOCKET(ZMQ_REP)
3: ZMQ_BIND(s, url) ▷ binds, should be run before connect
4: **while** $msg \leftarrow$ RECV_MSG(s) **do**
5: SEND_MSG(s, msg)
6: **end while**
7: CLOSE_SOCKET(s)
8: **end procedure**
9: **procedure** LATENCY_TESTER($url, msg_{size}, msg_{count}$)
10: $s \leftarrow$CREATE_SOCKET(ZMQ_REQ)
11: ZMQ_CONNECT(s, url)
12: $t_0 \leftarrow time_{now}$
13: **while** $i < msg_{count}$ **do**
14: SEND_MSG(s, msg_i)
15: $msg \leftarrow$ RECV_MSG(s) ▷ blocking receive
16: $assert:\ msg_i = msg$
17: **end while**
18: CLOSE_SOCKET(s)
19: $elapsed \leftarrow time_{now} - t_0$
20: $latency_{ms} \leftarrow elapsed\ /\ (msg_{count} * 2)$
21: **end procedure**

Algorithm 2, on the other hand, describes the procedure for measuring the throughput. Throughput is defined as the capability of the design to transport that number of bits in the form of payloads inside message packets. To measure the throughput performance, a PUB/SUB socket pair was used. The publisher hosted on the reference hardware continuously pushes messages with a configurable payload size. The subscriber on the tested node collects the messages as fast as its hardware allows and calculates the average throughput.

Algorithm 2 Message throughput assessment

```
1: procedure THROUGHPUT_TESTER(url, msg_count t)
2:     s ← CREATE_SOCKET(ZMQ_SUB)
3:     ZMQ_BIND(s, url)                    ▷ binds, should be run before connect
4:     msg_s ← RECV_MSG(s)                 ▷ blocks until first message
5:     t_0 ← time_now
6:     while i < msg_count do
7:         msg_i ← RECV_MSG(s)             ▷ non-blocking receive
8:         i++
9:     end while
10:    CLOSE_SOCKET(s)
11:    throughput_Mbps ← (msg_count * msg_size * 8) / elapsed
12: end procedure
13: procedure MSG_PUBLISHER(url, msg_size, msg_count)
14:    s ← CREATE_SOCKET(ZMQ_PUB)
15:    ZMQ_CONNECT(s, url)
16:    while i < msg_count do
17:        SEND_MSG(s, msg_i)
18:    end while
19:    CLOSE_SOCKET(s)
20: end procedure
```

3.6.3 Middleware node on a x86 architecture machine

The firsts stage of validation involved a regular x86 architecture for evaluating of the full range of functionality, validating the architecture of the middleware and its successful communication with the BMS. Furthermore, the performance results on this strong hardware sets the record and the reference for follow-up comparisons with the rest and more suitable architectures in the following subsections.

Performance evaluation

To begin with, Table 3.1 lists the RTT and maximum achievable throughput. The iPerf3 performance of a 1000BASE-T switched Ethernet servers as the absolute reference for the validation section as it is the most performing combination of network and hardware. In fact, the first row (non VPN) serves as the reference performance of the network infrastructure. However, all the middleware nodes are not expected to be wired. Thus, the table presents in addition the same performance metrics over Wi-Fi 802.11n links.

Table's column TCP C_to_S is the throughput from the test (Client) to the reference (Server) machine, while TCP S_to_C denotes the reverse direction. The dedicated tests per link direction is more applicable to the rest of the hardware platforms; due to their RF-hardware design and computational capabilities limitations, reception and transmission do not perform equally.

Table 3.1 also illustrates the differences in performance between a non VPN communication, two VPN ones using AES-128-CBC and AES-256-CBC cipher with LZO data-stream compression, and a VPN connection lacking the data-stream compression.

The reason behind those benchmarks is to assess the effect of the encryption complexity, computational compression overhead, and overall the effect on the VPN layer on the raw performance of the middleware hardware. While those tests are not presenting the middleware performance, they are useful for referencing and evaluating the behavior of the middleware that operates over those networks.

Table 3.1 – RTT and iPerf3 measurements on the maximum achievable bandwidth between the Intel® Core i5-5300U machine (client C) and the reference hardware (server S).

Link type	RTT	TCP C_to_S	TCP S_to_C
802.3 1Gbps			
No VPN	0.148 *ms*	934 *Mbps*	945 *Mbps*
AES-128-CBC cipher + LZO	0.636 *ms*	879 *Mbps*	869 *Mbps*
AES-256-CBC cipher + LZO	0.576 *ms*	839 *Mbps*	715 *Mbps*
AES-256-CBC cipher	0.635 *ms*	876 *Mbps*	889 *Mbps*
802.11n 300Mbps			
No VPN	0.84 *ms*	171 *Mbps*	210 *Mbps*
AES-128-CBC cipher + LZO	1.23 *ms*	165 *Mbps*	145 *Mbps*
AES-256-CBC cipher + LZO	1.23 *ms*	158 *Mbps*	120 *Mbps*

Table 3.2 – OpenSSL cryptographic ciphers performance on Intel® Core i5-5300U CPU. *Bigger is better, in KB/sec.*

	Block size				
Cipher	**16 B**	**64 B**	**256 B**	**1024 B**	**8192 B**
DES-CBC	64234.95	66960.94	67590.62	68051.16	67893.94
DES-EDE3-CBC	25466.24	25785.23	25835.28	25877.55	25961.76
RC2-CBC	42807.51	43872.77	44123.47	44261.93	44195.03
BF-CBC	104741.28	112803.18	114666.27	115759.33	115826.90
AES-128-CBC	110153.12	121826.92	124652.36	122825.73	124552.36
AES-192-CBC	88292.13	99244.99	103397.56	104542.14	104439.26
AES-256-CBC	81587.32	87219.84	89016.90	89443.40	89449.51

Table 3.2 presents the cryptographic cipher performance for the Intel® machine that is obtained using the OpenSSL library. The results are given in KB/sec for encrypting data blocks of 16, 64, 256, 1024, 8192 B. Moreover, the tests are run in single-threaded processes to better match

the single-threaded process of OpenVPN server and to facilitate the comparison with the single-core architectures. The table facilitates the understanding of the encryption cipher impact over the VPN tunnel. Additionally, as it will be proven when the other architecture tests will be introduced, there is not a universally bad or good cipher, performance-wise. It highly depends on the machine architecture and the existence or not of a cipher hardware acceleration. For example, authors in [183, 184] rank Blowfish (BF) as the best performing cipher, followed by the DES and the AES finishing last. However, Table 3.2 pictures a different situation on a AES hardware-accelerated core. The AES cipher has comparable performance to the Blowfish and surpasses the DES, despite that the latter is using smaller encryption keys. Moreover, despite the difference in key size in the three AES ciphers, due to the hardware acceleration, a doubling of key size does not considerable impact the performance.

The observations of Table 3.2 are validated in fact also by the previous Table 3.1. Increasing the cipher's key size from 128 to 256 bits did not considerably reduce. As a matter of fact, the performance degradation is observed immediately with the introduction of the VPN layer. For example, VPN connection over the 1Gbps link reduced the throughput to nearly half compared to the non VPN variant. Nerveless, with the more restricted bandwidth over Wi-Fi, the effect of the VPN is much less pronounced. Moreover, the high-speed compression algorithm enhances the throughput with minimal latency impact. The positive effect of the compression is better observed in the following figures, where the middleware messages benefit considerably from such data-stream compression.

Fig. 3.15 illustrates the measured latency across a REQ/REP socket pair. The measurements are collected using the Algorithm 1. The horizontal lines denote the reference raw latency as measured during the RTT tests, cf. Table 3.1. Since the message latency is measured per direction, $\frac{RTT}{2}$ is used in this figure. The Y-axis denotes the latency in microseconds on a logarithmic scale, while the X-axis denotes the message's payload size. There is not an absolute limit on the message size. However, a range of 5 B to 100 KB was considered wide enough for any particular middleware application within SB scope. In fact, messages of sensors and actuators range is on average around 50 to 250 B, including the middleware-specific data encoding and formatting. The figure plots the measurements for four different connections: Ethernet non-encrypted, Ethernet encrypted, Wi-Fi non-encrypted, and Wi-Fi encrypted.

One can observe in the figure that for payloads as large as 50 B, the latency is not more than the reference one in the raw connection. As a matter of fact, the latency of payloads $\leq$ 10 KB for Ethernet and $\leq$ 1 KB for Wi-Fi, traveling in a VPN tunnel, is even less than the reference latency. This is explained by the fact that ZeroMQ keeps the TCP connection alive between messages and thus eliminates the connection overhead which is more noticeable in high latency networks or through the VPN. Nevertheless, even for the maximum payload of 100 KB, the latency is acceptable for the time domain of building operations.

Fig. 3.16 plots the message throughput in Mbps versus payload size for the same four connections. The throughput is measured using a PUB/SUB socket pair, at the subscriber side.

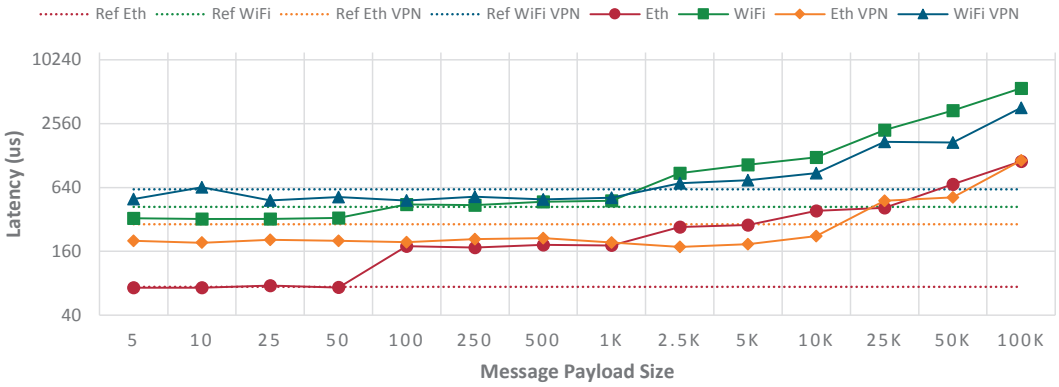


Figure 3.15 – Measured message latency on the x86 architecture.

It counts how fast the subscriber can successful receive and interpret the incoming message, cf. Algorithm 2. The subscriber node may aggregate many publishers in a 1-to-n manner, thus, the performance of the subscriber is important to be evaluated. As expected due to the tunnel overhead, the message throughput for both VPN enabled Ethernet and Wi-Fi is reduced, compared to the unencrypted ones. However, an interesting behavior can be seen on the message throughput on the VPN and it is noticeable on the Wi-Fi enabled VPN. The message throughput surpassed even the maximum TCP throughput as measured with the iPerf3 tool, cf. Table 3.1. To the author's opinion, this is the effect of the data-stream compression algorithm of the tunnels and the high compressive ratio of the middleware messages. The latter is the due to the standardized message encoding and the sharing of a significant portion of those metadata between subsequent messages. Finally, generally for payloads ≥ 100 B the throughput over individual messages reaches or even surpasses the absolute maximum achievable of the given connection, highlighting the zero-overhead aspect on the proposed middleware design.

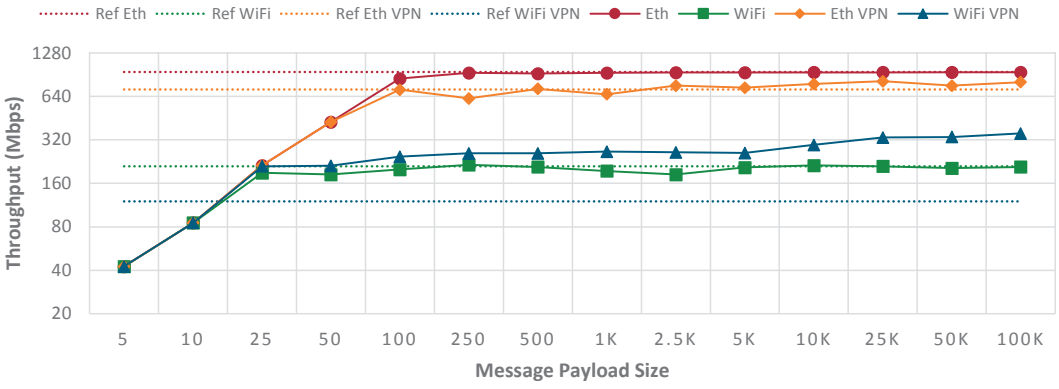


Figure 3.16 – Measured message throughout of a subscriber node on the x86 architecture.

Fig. 3.17 and Fig. 3.18 illustrate the effect of the programming language on the latency and throughput of the middleware respectively. This evaluation is critical because while the middleware nodes for performance evaluation were designed purely in C++, the middleware

nodes, developed for proof of concept into an actual building, were written in Python. Justification for that was the reduced development time in the scope of this dissertation for rapid usability and functionality evaluation. Python have improved considerably compared to past but still due to its dynamic nature, it is noticeably less computationally efficient. Despite that, the reader can observe in the figures that the latency is not affected by the particular language. Similarly, its throughput is only inferior for the smallest of the payloads, where a large number of messages need to be exchanged and Python source code to be looped.

In reality, the similar performance of both Python and C++ implementation of middleware nodes is easy to explain. While Python interpreter is relatively slow, the Python-ZeroMQ library is simply a language binding to a compiled version of the socket library. Therefore, the proposed middleware design, the communication library, and the various language bindings serve equally well the requirements of adaptability and expandability without sacrificing the performance.

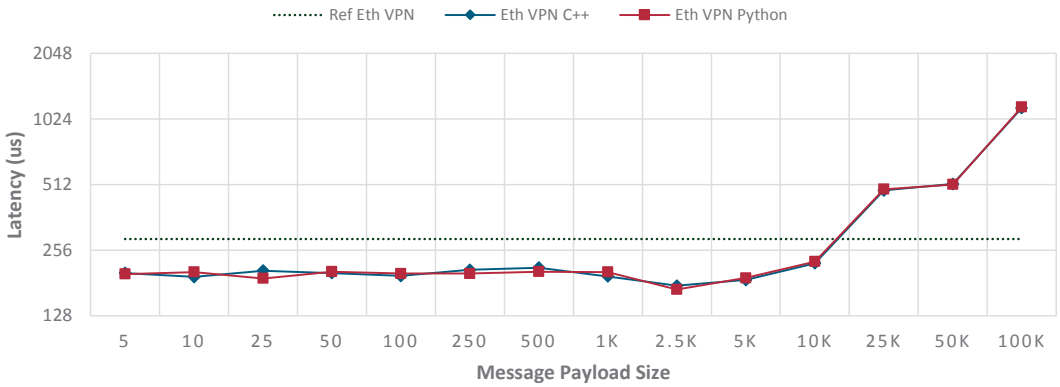


Figure 3.17 – The effect of programming language on the message latency on the x86 architecture over a VPN tunnel using AES-256-CBC cipher and LZO compression.

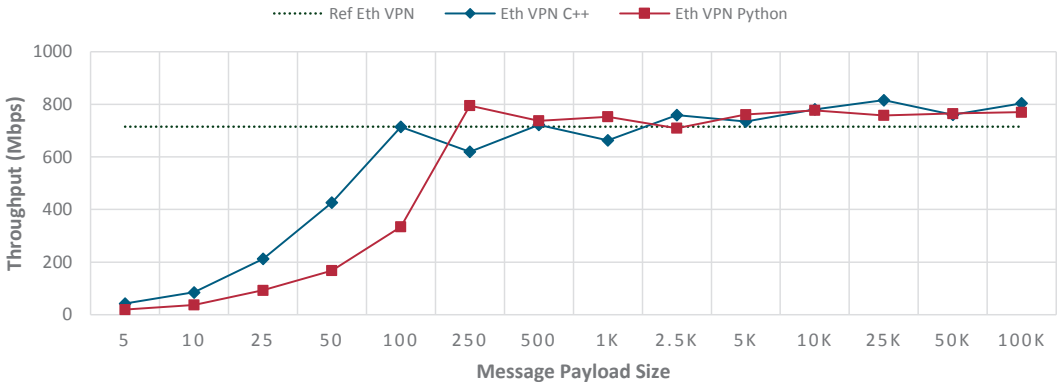


Figure 3.18 – The effect of programming language on the message throughput on the x86 architecture over a VPN tunnel using AES-256-CBC cipher and LZO compression.

3.6.4 Middleware node on an ARM architecture machine

The BeagleBone Black (BBB), as a candidate ARM® architecture machine, was considered as an embedded, reduced cost and yet capable enough, alternative to the x86 machine. Despite the embedded design, it is capable or running any Linux software compiled for the particular architecture. Moreover, a hard-float ABI ARM (armhf) kernel was used to enable the integrated hardware floating point unit and instructions of the MPU. Finally, the version of Linux kernel has been shown to contribute significantly to the performance of such embedded systems. Thus, the test system uses the most recent kernel based on the Texas Instruments codebase, version 4.9.13 revision 24 (4.9.13-ti-r24).

Performance evaluation

To begin with, Table 3.3 illustrates the raw TCP communication performance of the embedded board for both wired and wireless communication. Additionally, the performance of VPN communication using different ciphers and compressions is also presented. The reader could observe, that unlike the powerful x86 machine, cf. Table 3.1, the overhead of secured communication is much more pronounced, especially in the wired communication. The throughput becomes nearly one-third of the unencrypted ($\approx 94 \longrightarrow \approx 35$) and the RTT increases five-fold ($\approx 0.38 \longrightarrow \approx 1.97$). Yet, for the smaller throughput of wireless connection, the MPU can meet the computational demands of encryption layer without a drop in throughput performance (even an small increase due to the compression) and only with a slight increase in latency.

Table 3.3 – RTT and iPerf3 measurements on the maximum achievable bandwidth between the ARM® Cortex A8 machine (client C) and the reference hardware (server S).

Link type	RTT	TCP C_to_S	TCP S_to_C
802.3 100Mbps			
No VPN	0.38 *ms*	93.6 *Mbps*	94.1 *Mbps*
AES-128-CBC cipher + LZO	1.627 *ms*	31.2 *Mbps*	34.3 *Mbps*
AES-256-CBC cipher + LZO	1.97 *ms*	29.5 *Mbps*	34.7 *Mbps*
AES-256-CBC cipher	1.61 *ms*	31.4 *Mbps*	34.1 *Mbps*
802.11n 75Mbps			
No VPN	9.77 *ms*	9.77 *Mbps*	10.1 *Mbps*
AES-128-CBC cipher + LZO	12.4 *ms*	12.1 *Mbps*	13 *Mbps*
AES-256-CBC cipher + LZO	12.18 *ms*	10 *Mbps*	12.7 *Mbps*

Table 3.4 is the cryptographic cipher performance for the ARM® architecture similar to the X86 one, cf. Table 3.2. The procedure to obtain it is the same. The observations are also valid with this hardware architecture. Despite the complexity, the AES-based algorithms

seem to outperform their counterparts, while the impact of the key length on performance is smaller than the expected. Table 3.3 on the actual TCP throughput confirms the cryptographic performance, too.

Table 3.4 – OpenSSL cryptographic ciphers performance on ARM® Cortex A8 architecture. *Bigger is better, in KB/sec.*

	Block size				
Cipher	**16 B**	**64 B**	**256 B**	**1024 B**	**8192 B**
DES-CBC	15754.81	17847.52	18172.07	18336.11	18337.48
DES-EDE3-CBC	6363.79	6581.06	6632.28	6624.60	6652.23
RC2-CBC	12393.01	13257.12	13502.93	13558.24	13575.71
BF-CBC	25976.29	29053.20	30124.44	30395.33	30469.31
AES-128-CBC	36864.23	42537.38	44559.93	44950.19	45209.43
AES-192-CBC	31715.56	35657.44	37104.33	37452.37	37592.79
AES-256-CBC	28478.60	31579.52	32770.83	33050.88	33105.00

Fig. 3.19 illustrates the measured latency across a REQ/REP socket pair on the ARM® architecture. The process to collect those values is the same as before. The findings, although similar, have some differences. Analogous to the x86 hardware the latency is acceptable in the time domain of interest. The latency of payloads ≤ 2.5 KB over Ethernet and in the VPN tunnel, is even smaller than the reference RTT tests. In fact, even the unencrypted Ethernet exhibits same behavior for payloads ≤ 50 B. On the contrary, the messages through the wireless link, show greater latency, especially as the payload increases.

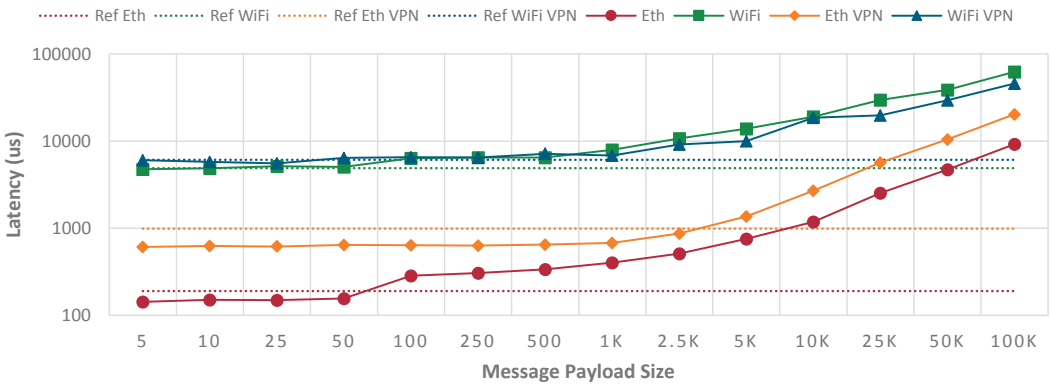


Figure 3.19 – Measured message latency on the ARM® architecture.

Fig. 3.20 plots the message throughput in Mbps, versus payload size for the same four connections, using a PUB/SUB socket pair similar to before. It is interesting to observe that while the message throughput is notably smaller than the one of x86 platform, the messages through the encrypted and compressed channel can outperform the reference TCP throughput.

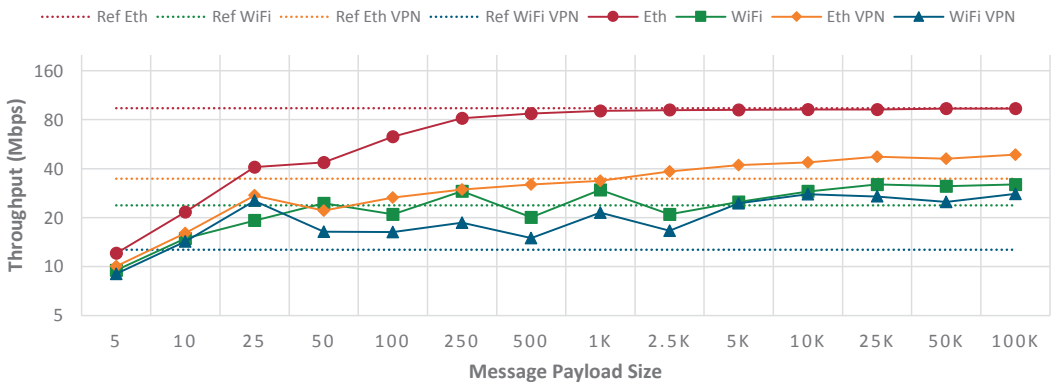


Figure 3.20 – Measured message throughout of a subscriber node on the ARM® architecture.

Fig. 3.21 and Fig. 3.22 illustrate now the effect of the programming language on the latency and throughput of the middleware respectively. This time, the Python creates a small but noticeable increase in the message latency. Additionally, the throughput performance is decreased by 10 Mbps or more compared to pure C++ implementation.

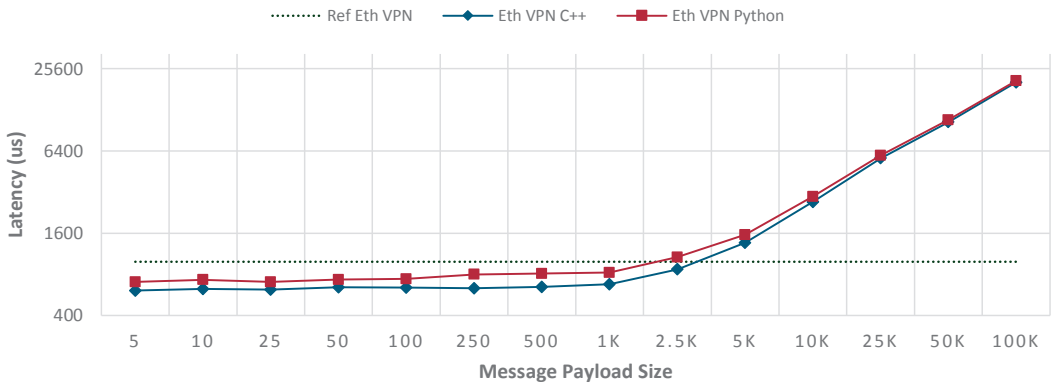


Figure 3.21 – The effect of programming language on the message latency on the ARM® architecture over a VPN tunnel using AES-256-CBC cipher and LZO compression.

Regardless of the physical network and encryption, this low cost, size, and power board can achieve and surpass the middleware design requirements. Its message, latency and throughput, performance is by adequate for sensible communication during regular and emergency building operations.

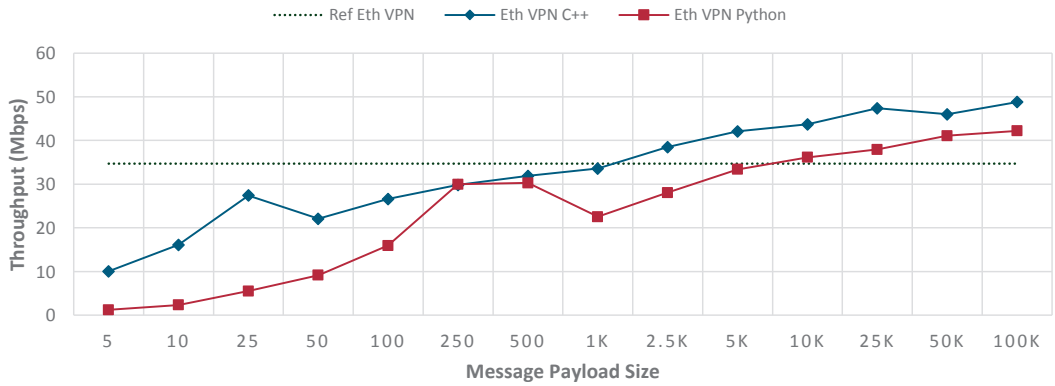


Figure 3.22 – The effect of programming language on the message throughput on the ARM® architecture over a VPN tunnel using AES-256-CBC cipher and LZO compression.

3.6.5 Middleware node on a MIPS architecture machine

LinkIt Smart 7688 Duo as a hardware platform is of particular interest for its extreme low cost and power while maintaining a high level of functionality. The SB designer can thus create a fully distributed middleware using a large number of such open-source hardware for the nodes. Different network interfaces, micro-databases, micro-controllers, and other management and intelligent services can be distributed within the building and across heterogeneous networks.

The research revealed significant advantages of such MIPS MPU architecture for meeting the demands and motivation of the middleware. For example, this small MIPS core is not only very efficient but also powerful enough to network live stream a USB camera feed.

Additionally, unique configurations and software libraries have been implemented during that time for the adaptation of such embedded micro-OS and the successful compilation of all the node software to the proper MIPS architecture, thus supporting the full functionality of the node communication layer.

Most of the test and adaptation software development has been implemented on the LinkIt Smart 7688 Duo platform. Nevertheless, the implemented codebase could be reused for any MIPS-enabled embedded electronics system that supports the OpenWRT micro-OS. The choice of LinkIt Smart 7688 Duo as a representative of the MIPS family of devices was justified because of its low cost, wireless connectivity, a large number of externally accessible GPIO, microSD interface, sufficient RAM, and integrated MCU. The MCU can take over all the real-time and time-critical processes of the protocol layer, Fig. 3.8, while the MPU and micro-OS handle the high-level communication, protocol adaptation, and any distributed intelligence. Finally, as this device is open-source hardware, any ICT systems designer can easily modify and extend it according to their needs.

Preparation of the LinkIt Smart 7688 for the tests

While for the first two hardware the setup preparation was pretty straightforward, the MIPS MPU, the limited RAM, and the restricted capabilities of OpenWRT make the initial setup deployment a challenging process. However, the cost versus the offered real-time performance and physical connections outweighs the burden of developing a middleware node on a MIPS MPU architecture.

Compiling any software, let alone a full library like the ZeroMQ, on Linkit Smart requires additional tools, configurations, and a long period of experimentation. Unlike x86-64 and ARM® machines that support native compiling of source code, there is neither enough memory and disk space nor processing power to install a building environment and compile the code natively; a "cross-compile" of the code on a computer is necessary. The "cross-compiling" building environment suitable for this architecture is the OpenWRT SDK. The SDK is enabled through a list of Makefiles and patches to create the toolchain for a given MPU architecture. The toolchain is a set of linked development tools for performing complex software development tasks. It is then used for building the software code and together with additional scripts to create the firmware OS image of the embedded device.

For compiling a single source file without dynamic libraries, the process is relatively easy. Firstly, the path of the compiled toolchain binaries that match the desired architecture should be added to the `PATH` environment variable. Secondly, the `STAGING_DIR` variable should point to the path of the toolchain directory. Finally, `mipsel-openwrt-linux-gcc` for C code, and `mipsel-openwrt-linux-g++` for C++ compilers can be used for building the source code.

The situation is far more complicated when multiple source code files need to be compiled and linked against other libraries. In this case, a Makefile defines the procedure to be taken. It takes into consideration the dependencies on third-party libraries, the multiple source and header files, as well as any dynamic and statically compiled libraries. The latter approach was used for developing the benchmark software of this section and integrating the ZeroMQ library (.so and .h files) into the OpenWRT firmware image.

The auxiliary tools, for example, the iPerf3, the OpenVPN client and the Python interpreter, were also compiled and integrated. Unlike the libraries and tools, the actual middleware node sources are kept separate from the firmware image and in the user space. This enables continuous updates to the node software without the need to compile and flash a full embedded OS image.

Concluding, cost-, connectivity-, and functionality-wise the MIPS based board is indisputably an excellent fit for the requirements of the middleware designed for SB. The subsection continues with studying, the performance, and energy efficiency potential, for an exhaustive assessment of this promising software-hardware synergy.

Performance evaluation

Table 3.5, much like the previous two hardware platforms, provides the reference communication performance of the evaluated device. Right away, one can observe two things. Firstly, the non-encrypted wired and wireless communication is superior to the more expensive ARM® board. Secondly, the encrypted tunnels have a significant performance impact on this less powerful hardware; the throughput is up to 7 times less on an Ethernet connection. Reducing the cipher key length only insignificantly improves the throughput performance. The real-time compression algorithm has no adverse effect on the TCP throughput despite the less capable MPU. The latency is tripled similarly to the ARM® one. Nonetheless, the throughput and latency are adequate for the needs of a middleware node under realistic communication activities. However, for the data concentrators and intelligence nodes, a more capable hardware is necessary; as the MT7688 and the 24KEc in general, lack a floating point unit hardware.

Table 3.5 – RTT and iPerf3 measurements on the maximum achievable bandwidth between the MIPS architecture (client C) and the reference hardware (server S).

Link type	**RTT**	**TCP C_to_S**	**TCP S_to_C**
802.3 100Mbps			
No VPN	0.43 *ms*	94.3 *Mbps*	94.1 *Mbps*
AES-128-CBC cipher + LZO	1.58 *ms*	13.9 *Mbps*	14.5 *Mbps*
AES-256-CBC cipher + LZO	1.58 *ms*	12.8 *Mbps*	13.6 *Mbps*
AES-256-CBC cipher	1.56 *ms*	12.9 *Mbps*	13.6 *Mbps*
802.11n 150Mbps			
No VPN	2.49 *ms*	24.7 *Mbps*	42.1 *Mbps*
AES-128-CBC cipher + LZO	3.22 *ms*	11.1 *Mbps*	11.7 *Mbps*
AES-256-CBC cipher + LZO	3.35 *ms*	9.8 *Mbps*	11.3 *Mbps*

Table 3.6 illustrates the cryptographic cipher performance of this MPU. The reader also can confirm the findings of researchers [183, 184] who ranked Blowfish (BF) as the best performing cipher. However, given the complexity of AES and the limited core, the performance of AES ciphers stands out. That is because this specific MT7688 MIPS core has an integrated AES-128/256-CBC encryption engine, accelerating the cryptographic process. Due to only minor differences in measured TCP throughput, cf. Table 3.5, the AES-256-CBC cipher was also the best choice for the encryption and performance needs.

Fig. 3.23 illustrates the measured message latency, for the MIPS® architecture using Algorithm 1. In this case, especially for the encrypted communication, the latency increases considerably for large payloads. However, for payloads ≤ 1 KB, the message latency remains close to the reference RTT measures. Moreover, the reader can observe that even with this less powerful MPU architecture, the latency remains reasonable even for payloads as large as 100 KB. Thus,

for the most probable payload sizes and despite the limited MPU, the Linkit Smart 7688 can effectively address the latency requirements.

Table 3.6 – OpenSSL cryptographic ciphers performance on MIPS architecture. *Bigger is better, in KB/sec.*

	Block size				
Cipher	**16 B**	**64 B**	**256 B**	**1024 B**	**8192 B**
DES-CBC	4830.23	4906.68	4902.96	5034.08	4967.78
DES-EDE3-CBC	6363.79	6581.06	6632.28	6624.60	6652.23
RC2-CBC	4847.14	5014.49	5082.84	5110.03	5076.31
BF-CBC	8739.50	9482.95	9703.52	9684.85	9657.79
AES-128-CBC	7253.09	7887.45	8102.61	8122.03	8148.16
AES-192-CBC	6331.75	6823.83	6987.26	7044.30	7006.89
AES-256-CBC	5603.30	6025.31	6139.72	6189.36	6144.00

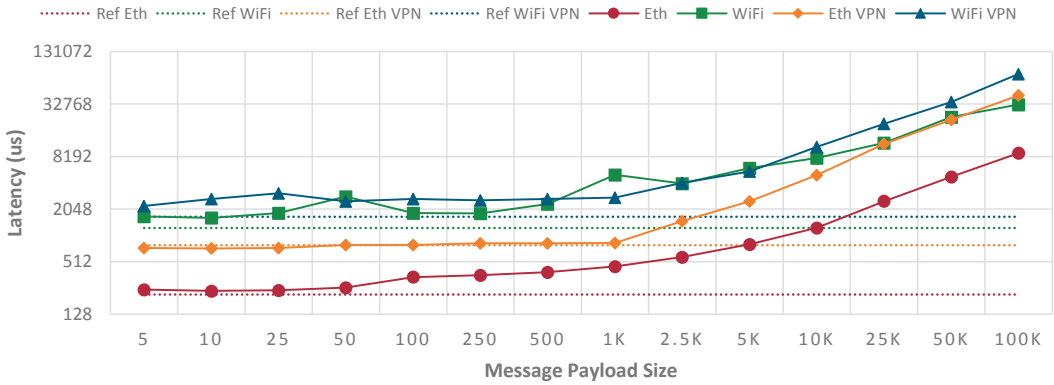


Figure 3.23 – Measured message latency on the MIPS architecture.

The throughput capabilities of such hardware are plotted in Fig. 3.24. The expected encrypted tunnels performance degradation is observed on the message-oriented tests. On Ethernet the payload through VPN are only able to transport at least one order of magnitude compared to unencrypted transfer. Nevertheless, unless it is a critical data routing or a concentrator node, the encrypted throughput of down to 16 Mbps (wireless) is more than enough.

Finally, Fig. 3.25 and Fig. 3.26 demonstrate the effect of programming language on latency and throughput of the middleware respectively. In that regard, Python increases the message latency slightly compared to native C++ implementation. On the other hand, the throughput is severely reduced for most payload sizes. The author believes that while a Python-developed middleware node will still remain functional, the performance impact on this hardware is significant enough to recommend a native alternative. However, the latter has a significant drawback. Since the MIPS platform is unable to compile the code natively, for every software update, the cross-compiler environment should be used. Interpreted Python code, on the

other hand, needs simply a file update.

Concluding, it is clear that a MIPS-enabled board cannot match the previous ones. However, the reduced cost and energy use and the relaxed communication performance requirements fully justifies it as a middleware node hardware.

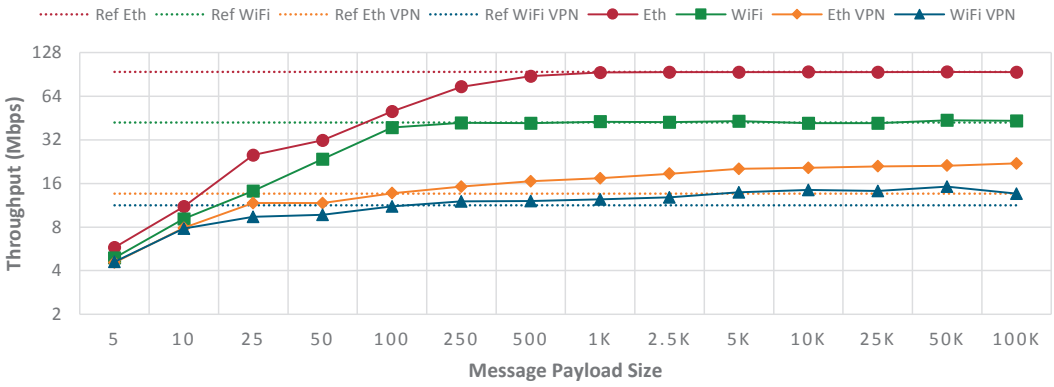


Figure 3.24 – Measured message throughout of a subscriber node on the MIPS architecture.

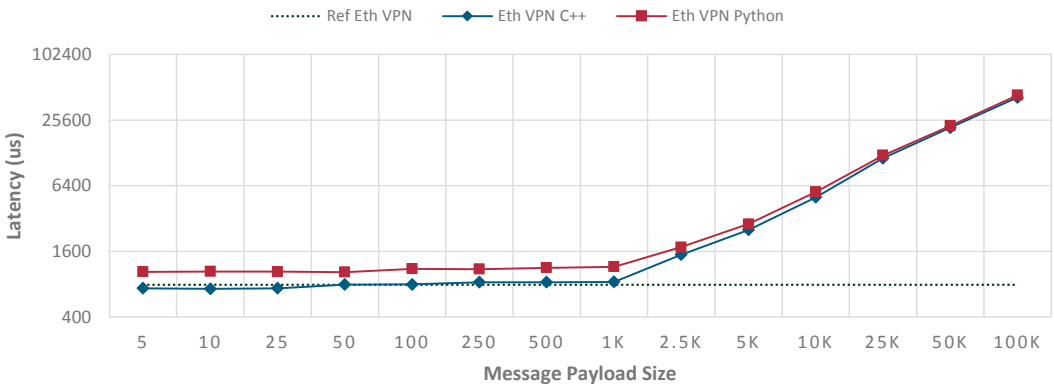


Figure 3.25 – The effect of programming language on the message latency on the MIPS architecture over a VPN tunnel using AES-256-CBC cipher and LZO compression.

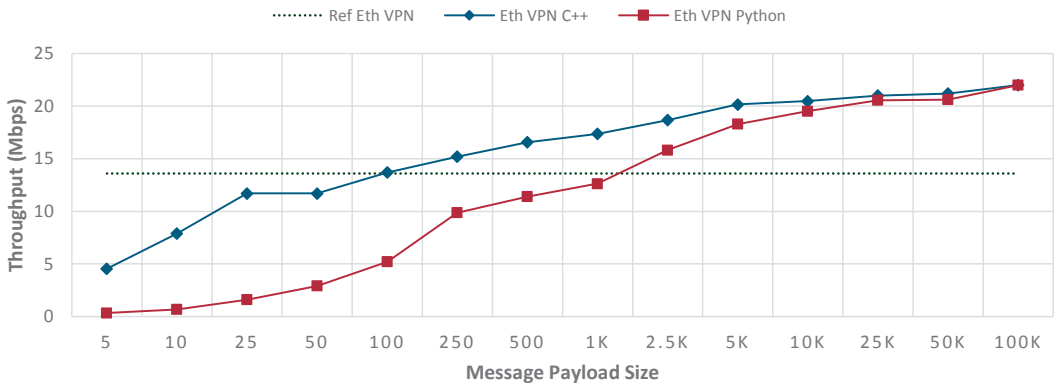


Figure 3.26 – The effect of programming language on the message throughput on the MIPS architecture over a VPN tunnel using AES-256-CBC cipher and LZO compression.

Energy evaluation

To measure accurately and in real time the current used by the module, an ultra-high accuracy (0.1% max gain error and 10 uV max offset) current monitor chip was used. This type of device is frequently found in embedded designs that require real-time power consumption data. More specifically, the device records the shunt voltage of a 0.1 Ohm resistor; together with the 16-bit resolution of its ADC, it yields a current least significant bit (LSB) of 0.025 mA and a maximum measurable current of 819.2 mA.

To begin with, Fig. 3.27 illustrates the current use on receiving a burst of 100k messages of 500 B, without VPN and using the Ethernet connection. The test also includes the loading time of the compiled binary from the flash memory as illustrated between the two first markers, taking ≈ 0.4 sec. The time between the last two markers denotes the burst reception time, the calculated throughput of 85.1 Mbps confirms the previous tests. It illustrates the real-time current consumption during the high activity reception, averaging 160 mA.

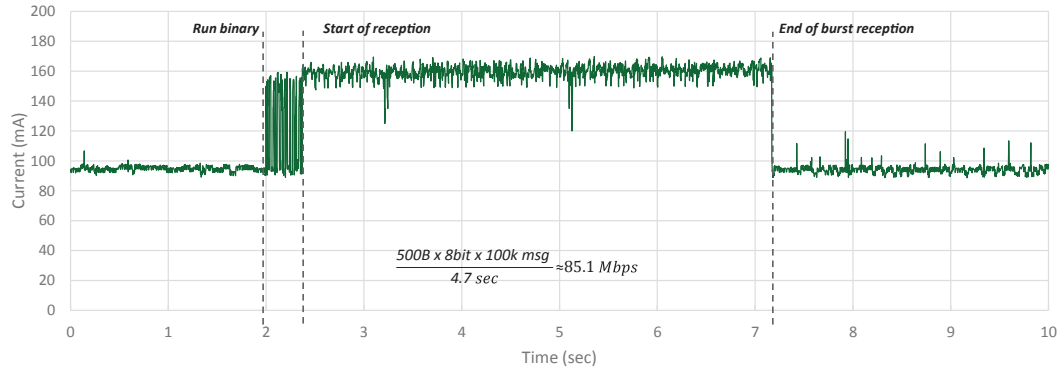


Figure 3.27 – Power consumption of Linkit Smart 7688 Duo during the wired and unencrypted reception of 100k, 500 B messages using a natively compiled binary (C++) and a SUB socket.

Fig. 3.28 is very similar to the previous one, except a wireless instead of a wired connection and

the number of received packets. The reception lasts ≈ 4.2 sec with a calculated throughput of 43 Mbps. The current consumption during that time is higher than before, on average 190 mA.

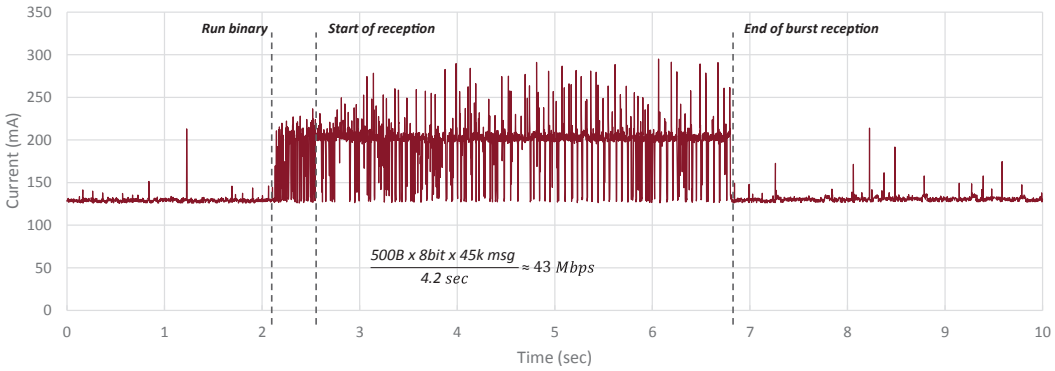


Figure 3.28 – Power consumption of Linkit Smart 7688 Duo during the wireless and unencrypted reception of 45k, 500 B messages using a natively compiled binary (C++) and a SUB socket.

Finally, Fig. 3.29 reflects the effect of Python on the energy use by this particular middleware node. An interpreted language like Python requires far more resources compared to compiled code. While performance-wise the impact is acceptable when it comes to energy use, the results are far from ideal. Unlike the fast startup of the compiled binary, the Python script requires a preliminary loading of the Python interpreter, a demanding process for this small core. Moreover, even if the script is loaded only once, each line of code requires more processor cycles and thus consumes more energy. Thus, a Python-based middleware node, although desirable extendability-wise, when it comes to energy use, the compiled languages should be preferred. This particular figure illustrates the energy consumption during wireless transmission (PUB socket) of 20k messages of 500 B.

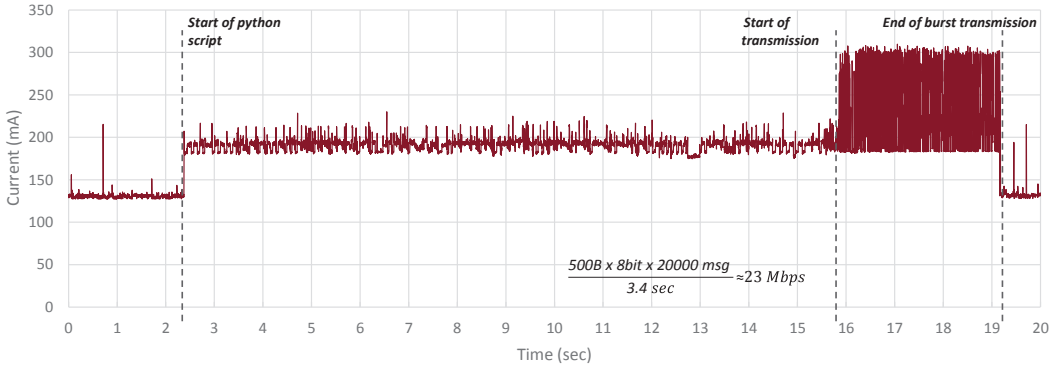


Figure 3.29 – Power consumption of Linkit Smart 7688 Duo during the transmission of 20000 500 B messages using Python and a PUB socket over a wireless connection and without VPN.

Unfortunately, the amount of energy required to operate this module is far more when considering battery-only deployments. However, since the middleware node may have to control critical loads and monitor environmental data, it should remain functional despite any power supply disturbances. Thus, the potential operation under a backup battery was

evaluated in order to maintain the critical functionality, therefore improving the fault tolerance of the system despite the environmental conditions.

Two approaches are generally available for achieving energy-efficient operation under a constrained power budget. Firstly, the passive approach involves energy efficient hardware such as this MIPS board, compiled binaries, and deactivation of any unused communication interfaces at the kernel level. Secondly, a more aggressive active approach required placing the MPU in low power mode and shutting down internal and external to the SOC peripherals.

Unfortunately, OpenWRT OS lacks proper power management support such as "suspend to RAM" (sleep mode). In order to conserve energy and artificially "suspend" the MPU, its power supply rail must be deactivated. In fact, this module has an interesting property. Thanks to the dedicated MCU, the artificial suspend and resume process can be automated.

In a prolonged interrupted power scenario, for example, the MCU continues to collect measurements from the interfaced network and temporarily store those raw values. Every time interval, it activates the MIPS MPU and transfers to the OpenWRT user space the vector of raw values using their physical UART connection. The MPU on its behalf will implement the necessary protocol adaptation and then communicate the collection of messages to the whole middleware as usual. After completing its task, it notifies the MCU to deactivate the MPU power rail again.

Concerning the message queue at the communication layer of the node, messages of the SUB socket are dropped when it is suspended. However, a REQ-REP pair queues the messages on the sender side, thus when the middleware node comes online it can serve all the requests in the receive queue. The previous fact is an additional reason for having two functionally different socket pairs.

Luckily, despite the lack of an integrated supply rail electronic switch, the reset pin of the MT7688 (PORST_N) is accessible. By connecting that to an available GPIO on the MCU, the latter can control the activation and deactivation of the MIPS core by keeping it in the reset state, which draws only minimal supply current. While it is not ideal, it is still functional without hardware modification.

As the MPU resets, the full booting up and initialization process energy and time should be taken into consideration. Fig. 3.30 plots the current used and the time required by the module during booting up and network initialization. The boot sequence can be seen with the plot markers denoting the important completed steps of the process:

- Bootloader: the bootloader from flash gets executed, performing the necessary low-level hardware initialization. Then it decompresses the kernel from the flash memory into the RAM. Finally, it executes the decompressed kernel image, passing it the pre-configured options.
- Kernel: the kernel also starts with hardware and low-level software initialization. Then

it mounts the read-only firmware partition (SquashFS). At this stage, the module can begin basic communications over Ethernet using network configuration burned into the read-only firmware image. Following the read-only partition mounting, the kernel mounts the rest of the now writable partition under "rootfs'. At this stage the custom network and module configuration are loaded in memory and available for initialization. Finally, it executes the *init* process, initializing the OS user space.

- Init: at this stage the wired and wireless network initialization begins according to the provided configuration. The OS executes the start-up scripts and brings up all the foreground and background services necessary for a fully functional OS. Preparing the network interfaces, device drives and setting up all the services is the most time-consuming part of the booting up process.

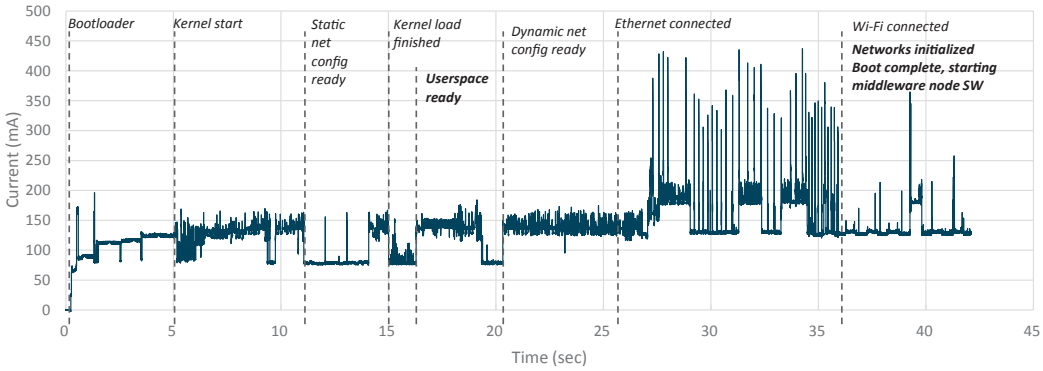


Figure 3.30 – Power consumption of Linkit Smart 7688 Duo during OpenWRT booting up and network initialization.

Fig. 3.31 illustrates that transition from the power-on phase to the reset phase initialized by the MCU pulling the PORST_N pin to 0. While in the reset state, the combination of MCU and MPU consumes much less energy. The figure features two plots. The highest current represents the wireless connectivity while the wired one not only operates with less current but also during reset it uses less than half of the wireless one.

The wired connection allows the board to operate from as low as 3.75V on the 5V input, suggesting an excellent fit for wiring it directly to a backup Li-ion secondary cell with a rated voltage of 3.8V, defeating the need for additional voltage regulator and hardware modifications. Alternatively, using the exposed 3.3V unregulated supply pin, an external high quality, low dropout power measurement unit (PMU) can be utilized. It can also drive the board from even lower input voltages (better utilizing the cell) and provide battery charging functionality. In fact, when supplied using the unregulated 3.3V rated input, the modules continues to communicate over Ethernet down to 2.85V, allowing it to utilize the energy of a secondary Li-ion cell completely.

Nonetheless, it is clear that even with the passive approach to energy efficiency, the module can maintain full functionality for a entire day even using a single Li-ion cell of 2000 mAh. With

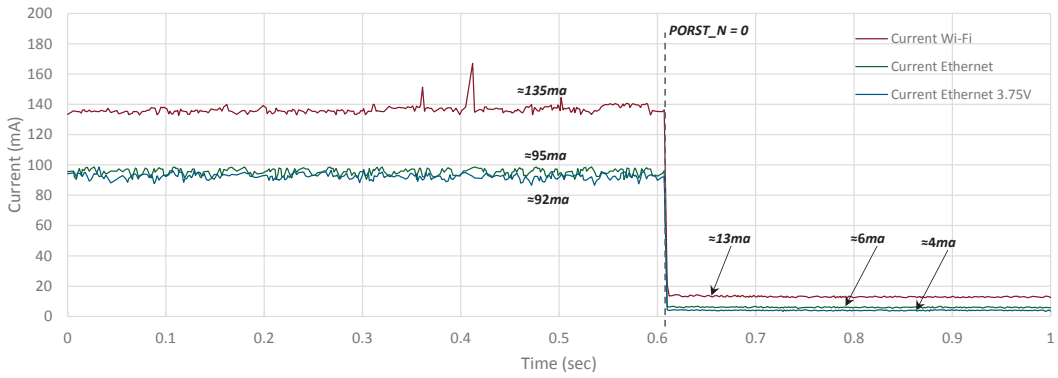


Figure 3.31 – Power consumption of Linkit Smart 7688 Duo during forced suspend using the reset (PORST_N) pin.

a more aggressive active approach to energy management, the autonomy extends to a few days of limited middleware functionality.

3.7 Conclusions

This chapter introduced the major challenges inhibiting the creation of a generic and scalable BMS. It then proposed a middleware-enhanced approach for addressing these difficulties. The middleware acts as an interface between the physical world, embedded devices, and networks and the high-level management and building intelligence services. The motivation and aims of such a middleware system were studied and used as input requirements for its design. Moreover, a comprehensive literature review proved that while some middleware systems exist, they are either not suitable for SB or they are not addressing the design requirements set by the author.

The proposed solution's unique characteristic is the distributed nature allowing the middleware to adapt to the physical features of the building, abstracting the BMS from the physical world particularities. The communication is based on the MoM approach for asynchronous communication and high decoupling between the distributed middleware nodes. The lack of a centralized middleware node (broker) increases the robustness and the speed of the system, while a centralized node directory addresses the nodes' self-discovery challenge. A VPN provides a security layer, offering a standardized and validated approach for securing all middleware communications. Finally, the chapter concludes with extensive performance and energy assessment of 3 unique hardware architectures for hosting the middleware node software.

4 Building-in-the-Loop Emulation Engine

Many aspects of information and communication technology (ICT) system and energy-related infrastructure of a Smart Building (SB) are not expected to be ready for implementation in the first few incarnations of the SB. However, both computer science and energy management literature already anticipate such infrastructure. This chapter proposes and validates a holistic software system based on discrete-event simulation (DES) for the SB real-time emulation. This engine enables the virtualization of building components and their transparent integration into the existing system while the building management system (BMS) remains agnostic to the virtual nature of the emulated infrastructure. Some unique features of the emulator include its micro-treading core and the high-performance communication layer based on asynchronous messaging. Those permit a highly optimized, concurrent emulation of hundreds of building elements (e.g., loads, batteries, generators, sensors, actuators, users, etc.) on commodity hardware, in real time. The modular design of the core ensures a scalable architecture, both regarding the type of emulated models and their number. The decoupled model running enables an on-demand emulation accuracy/performance adjustment. Candidate applications include the financial and energy gains evaluation using virtual infrastructure, high-speed simulation tools for the research and development of energy management algorithms, and finally, support of occupant-oriented behavioral studies.

4.1 Introduction

While the Smart Building (SB) is not a new notion to the building sector, recent developments uncovered features that grow beyond the automation domain. The new SB have become some of the most complex systems of interconnected information and communication technology (ICT) devices, energy-related elements, and various heterogeneous stakeholders.

Nevertheless, considerable steps have been taken towards technology interoperability for catalyzing their market adoption. However, the ICT systems are still a considerable investment upfront without guaranteeing improved energy performance or financial gains [7]. In order to improve these technologies and integrate services leveraging them, many simulation tools have been developed. These tools enable the user to estimate the performance of a building and analyze whether a retrofitted investment in passive or active SB technologies is justified.

The majority of these simulators are either designed to execute offline, in an ahead-of-time manner, or dedicated to specific aspects (heating, occupancy, etc.), which limits their application potential. They are frequently proprietary and expensive. On the other hand, literature proposes various solutions for building simulation; however, these tools are focused mainly on the modeling and accuracy innovations, while the optimized software implementation is frequently secondary. Because research primarily focuses on proof of concept studies, there is little available information relevant to the development of practical, large-scale simulators.

This chapter proposes a new building virtualization software solution. It is based on a custom emulation engine and simulation models for SB infrastructure, ICT systems, and occupants. The aim is for this tool is to be integrated into the existing building management system (BMS) [185], cf. Chapter 2, in order to provide real-time emulation capabilities, as seen in Fig. 4.1.

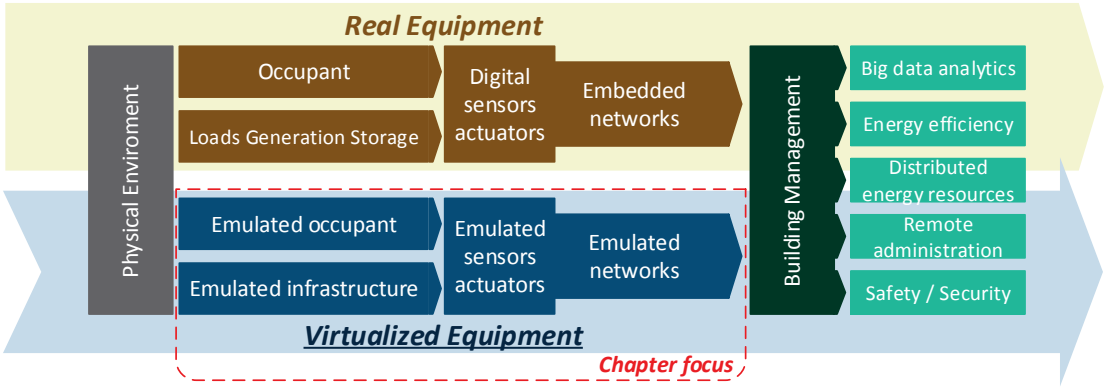


Figure 4.1 – The proposed building emulation engine as a virtualization technology, in parallel to, and integrated with, existing physical infrastructure.

The rest of this chapter is organized as follows. Section 4.2 describes the motivation for pursuing this research topic, Section 4.3 presents the state of the art in building simulation as compared with this solution, while Section 4.4 dives into the theoretical background and formulates the

theory on which this work is based. Section 4.5 scrutinizes the system architecture, detailed implementation and model aspects of the emulation engine. Finally, Section 4.6 assesses the performance as a virtualization solution and highlights it using a realistic case study. This chapter finishes with conclusions in Section 4.7.

4.2 Motivation

From the BMS point of view, the virtualization functionality is abstracted. Hence, the management system interfaces both real and virtual equipment using the common middleware, presented in Chapter 3. The proposed solution is to use the BMS application server to represent its emulated components with the existing data models presented in Chapter 2. Using this, the management and optimization algorithms continue to function without re-engineering when their physical counterparts in incremental investments replace the virtual components.

An advantage of this work, compared to literature, is the integrated, embedded network emulation seen in Fig. 4.1. The integration of the virtualization functionality into an existing BMS and its real-time requirements necessitates the creation of such network emulator system between the virtual devices and management server. The network module is emulating the constraints, such as packet losses and delays, that typically hamper the real physical monitoring and control devices. It empowers the incremental investment in real equipment while still maintaining the same BMS and intelligent services configuration.

The proposed work goes beyond studying the simulation models. It implements an optimized emulation engine based on high parallelism architecture, which features a micro-threading technology. This allows an emulation of each virtual component in a dedicated micro-thread. As a result, a large number of SB devices can be emulated in real-time, in low-cost commodity hardware.

Scientifically speaking, there is a strong interest in virtualization technology as a means to assess, optimize and validate advanced energy management algorithms without a prior investment in an inventory of costly infrastructure. With additional software modules, the proposed technology can also provide value to investment's financial analysis and energy performance estimation studies before committing to expensive building retrofitting. Consequently, financial gains can be estimated not only through energy efficiency but also with the support of demand side management (DSM), where utility activities in the Smart Grid (SG) interact with the SB's generation and storage services [186, 187].

This emulation engine is currently being used as a testbench for DSM research projects. Such projects are leveraging the potential of virtualizing complex and expensive energy generation and storage infrastructure without costly investments; the purchase would follow only after successful validation of the DSM algorithms. The adoption of this work as the "de facto" simulation mechanism by follow-up research project highlights the scalability and integration

advantages and asserts its emulation accuracy.

Finally, the support of user models in the virtualization engine allows occupant-oriented studies to test various scenarios of user behavior and its impact on the energy consumption. Multiple behavioral parameters could be tested ahead of time and the ones to yield the best results to be returned as accurate and personalized energy feedback.

4.3 State of the Art

The SB is classified as a cyber-physical system (CPS) [117, 118]. This implies a seamless synergy of both the physical and digital components in a mutual existence environment. The CPS domain scope exceeds the embedded systems of today, enabling improved security, safety, extended capabilities and scalability potential.

A shortcoming of the CPS in general lies on their complexity, not only during the design and simulation but also for their operation. Firstly, they interface two fundamentally different sectors of engineering, the physical and the digital one. Unlike the continuous time modeling of the physical systems, digital ones are modeled predominately in discrete forms and time scales. Additionally, digital sub-systems communicate concurrently over non-deterministic and frequently heterogeneous networks.

Hence, creating an entire CPS simulator is not a trivial task. Palensky in [188] discusses those challenges and methods involved. Moreover, Talcott in [189] documents the event-based semantics in the CPS context and highlights the challenges of a human-centric system. Some other solutions proposed in the literature include the generic co-simulation [190], and the energy-aware CPS simulator [191]. This work is comparable with those on the aspect of asynchronous event processing thanks to the discrete-event simulation (DES) driven architecture.

Researchers in [117] modeled a modern building entirely as an energy CPS for joint optimization of the energy use by its occupants and ICT equipment. The authors in [118] describe how the CPSs can provide rapid access to information and decision-making enabling buildings to interact with the SG autonomously. Moreover, the [192] presents a co-simulation toolchain with a case study on the heating system of a SB. Finally, [193] is a state of the art review for Matlab based simulation tools focusing on buildings and their HVAC systems. Such models and simulation tools had provided the inspiration and motivation of creating the presented emulation engine which also features high integration with the BMS.

A comparable with this work study is the [194]. The authors present a rather flexible residential energy simulator and scheduling setup. They enable the integration of both renewable energy resources and battery storage. Moreover, their simulated loads are considered "smart appliances" which allow rescheduling. Similar to that study, this work enables various interactive scenarios for the emulated building components. However, unlike that work with

its predefined datasets, the proposed one is integrated into an existing BMS, enabling realistic interaction with non-simulated physical infrastructure. More specifically, the data generated by this engine are displayed and stored in the BMS cloud infrastructure. Last but not least, the study mentioned did not investigate the software architecture for implementing such simulations. On the contrary, an optimized emulation engine of this work targets and achieves large parallelization of numerous components even on the weakest hardware.

Another related work is the context-aware simulation system for smart homes by the authors of [195]. Although their scope is also to virtualize the smart home devices like sensors and actuators; they do not implement virtualization of load schedules, behavior of users, energy generation, and storage installed in the building, unlike this work. Furthermore, similar to the previous literature mentions, no attempt was made for optimized engine design which enables real-time and continuous operation within existing building management infrastructure.

A service-oriented approach to building simulation is conducted by researchers in [196]. Their distinctive feature is the modularity of the open platform which allows different users to participate and contribute to the development. The service-like architecture permits an effective and simplified introduction of new services without entangling the developers with the complexity of the BMS. Their simulation design integrates both real and simulated devices like the architecture presented in this work; however, they only do so at the web service level. In addition, that work does not study the simulation models but solely defines the service-oriented framework, leaving the model design to the developers. On the contrary, this work proposes an optimized, yet scalable and expandable solution for integrating real devices along with simulated ones at the network, instead of the web layer. Additionally, it not only defines the models' architecture but also implements and validates a comprehensive list of models.

Although the studies mentioned above characterize and model the CPS and the SB as a whole, state of the art includes additionally dedicated virtualization approaches for sensors and actuators. Due to their low power design and limited network performance, they are mostly in suspend mode until an external or internal event triggers them. The authors of [197] modeled those events and created event virtualization services accessible through the Internet. Furthermore, the literature provides a couple of studies on creating virtual sensors. Firstly, Merentitis et al. in [198] defend the sensor infrastructure virtualization as the driver towards the evolution of Internet of Things (IoT). Additionally, a sensor oriented middleware with virtualization capabilities over UDP/TCP is presented in [146]. Finally, the wireless sensor network (WSN) virtualization can go as far as embedded battery storage simulation [199] or even emulation of the physical RF channel in body area networks [200]. Last but not least, the functional virtualization of sensors and actuators, like the one in this work, should not be confused with the WSN virtualization like the proposed one in [201]. The latter performs a logical abstraction of the physical computing resources which enables, for example, many embedded applications to share the same WSN.

Finally, the studies of multi-agent systems like the [202, 203, 53] are simulating based on the "actors" of an SB. Those can either be the occupants or the automation systems which interact and possibly compete in various ways. While this work includes a user virtualization functionality that can act based on modeled profiles and schedules; it is not focused on creating collaborative or competitive user-agent models similar to the ones in the multi-agent systems. On the contrary, each emulated component is responding to various events by changing states and internal activities.

Consequently, to the author's knowledge, it does not exist a virtualization solution that can emulate at the same time and platform, not only the energy driven infrastructure but also the occupants and the ICT systems. The tight semantic parallelism between the physical and the virtual objects enable their seamless cooperation under various ambient intelligence and management supervision as part of the BMS operations. Lastly, the majority of the modeling and simulation approaches in the literature seek and favor the accuracy rather than efficiency in the computational algorithms, limiting their potential for commercial building automation solutions.

4.4 Theoretical Background

Simulation is the procedure that recreates the behavior of a system under various condition and parameters, in continuous or discrete manner. System simulations fall into one of the following categories: DES, Continuous Simulation or Agent-Based Simulation. However, many implementations tend to be designed as a combination of those, hence defined as *hybrid* simulator. In such cases, continuous varying models coexist with discrete ones driven by events.

The proposed emulation engine follows a hybrid simulation approach based on DES. This section presents the mathematical background, the engine design foundation along with the computational technologies for realizing a highly concurrent, real-time emulation engine.

4.4.1 Real time discrete event system specification

The discrete event specification (DEVS) formalism was initially presented by [204] to mathematically describe any system whose models change states by reacting on discrete external or internal events. An extension of the former, the real-time discrete event specification (RTDEVS) associates an activity to each state of the model, due to its real-time execution [205]. Considering hierarchical event-based models that can be broken down into more basic ones, any entity is described by the real-time atomic model (RT AM) of the RTDEVS formalism as seen in Eq. 4.1.

The RT AM describes the transitions between internal states S due to incoming events X that produces output events Y. Whenever an event is externally received, δ_{ext} defines how the

states should change, while δ_{int} deals with state changes due to internal events. The time advanced function $ta(s)$ specifies the duration up to which an internal event can be triggered. When expired, the output function λ indicates the output that has to be generated. If an external event is received before the completion of $ta(s)$, the new state s', computed by δ_{ext}, sets a new $ta(s')$.

$$
\begin{aligned}
&RTAM = < X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta, ti, \psi, \mathcal{A} > \\
&where \begin{cases}
X: & \text{a set of input events} \\
S: & \text{a sequential state set of the model} \\
Y: & \text{a set of output events} \\
\delta_{ext}: & \text{an external transition function, } Q \times X \rightarrow S \\
 & \text{where } Q \text{ is the total state set of } M = \{(s,e) | s \in S \; and \; 0 \leq e \leq ta(s)\} \\
\delta_{int}: & \text{an internal transition function } S \rightarrow S \\
\lambda: & \text{an output function } S \rightarrow Y \\
ta: & \text{a time advance function, } S \rightarrow \mathcal{R}^+ \\
ti: & \text{a time interval function, } S \rightarrow \mathcal{R}^+ \times \mathcal{R}^+ \\
 & \text{where } ti(s)|_{min} \leq t(a) \leq ti(s)|_{max}, ti(s)|_{min} \leq ta(s) \leq ti(s)|_{max}, \\
 & s \in S, a = \psi(s) \in \mathcal{A}, and \, t(a) \text{is the execution time of an activity } a \\
\psi: & \text{an activity mapping function, } S \rightarrow \mathcal{A} \\
\mathcal{A}: & \text{a set of activities, } a | t(a) \in \mathcal{R}^+, a \notin \{X?, Y?, S=\} \\
 & \text{where } X? \text{ is the action of receiving data from } X, \\
 & Y? \text{ is the action of sending data through } Y, \\
 & \text{and } S= \text{ is the action of modifying a state in } S
\end{cases}
\end{aligned}
\tag{4.1}
$$

For the DEVS, time changes only when the $ta(s)$ function is called by the simulator. The RTDEVS formalism introduces more parameters aiming to enhance those time-advances with executable activities. On that regard, the function ψ maps a state to an activity $a \in \mathcal{A}$. The ta in the case of RTDEVS also verifies the correctness of the mapping and compensates for the non-deterministic events' time discrepancies. The latter is bounded by the time interval function ti which expresses the time range that the execution of the activity a, and the ta should respect.

The discrete-event based model consists of an interconnection of those basic RT AMs. A real-time coupled model (RT CM) interconnects the various events from each atomic models and other coupled models as seen in Eq. 4.2.

$$RTCM = < D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$$
$$where \begin{cases} D: & \text{a set of component names} \\ \{M_i\}: & \text{a set of component basic RTDEVS models, } \forall i \in D \\ \{I_i\}: & \text{a set of influences of } i, \forall i \in D \\ \{Z_{i,j}\}: & \text{the } i\text{-to-}j \text{ output translation, } Y_i \rightarrow X_j, \forall j \in I_i \end{cases} \tag{4.2}$$

The set D lists the names of the components M_i, which are either RT AM or RT CM. Their coupling is specified by the influences I_i and i-to-j output translation $Z_{i,j}$. There are three types of coupling specification. The first two external coupling types, connect the input or output events of the coupled model to one or more input or output events of its components respectively. The third internal coupling type connects the output events of the components to the input events of other components.

4.4.2 Building Emulation engine as a DES system

DES design approaches and principles

The subsection 4.4.1 studied the theoretical background and formalism on the models governing the discrete event systems. To become simulations, they require an algorithmic approach for their execution. The past was dominated by general purpose simulation-oriented languages like SIMULA and GPSS. Recent decades effort has focused towards enabling simulation architectures using conventional languages such as C++ [206], Java [207] and Python [208] using additional software libraries. Regardless of the language of choice, there are three main approaches for simulating the discrete event models [208, 209].

Firstly, in the *event-based* approach, the collection of events triggers state changes and generate follow-up events and actions to be executed. The event models specify that behavior of the system. The *event-based* approach is the simplest and one of the most commonly encountered. For example, a user activates a device or changes a setting and the system timely responds; thus this approach excels in modeling the non-deterministic behavior of a system.

Secondly, the *activity-based* approach simulates the system behavior as a collection of activities. Each activity model represents a time-consuming, specific action performed by that entity. The behavior is based on time schedules where activities start, last a given amount of time and end with appropriate actions in each phase. For example, the building heating using hot water radiators can be such an example of activity. With the activity start, the burner starts to warm the water. When the water reaches a configured temperature T_1, the flow in the heating elements begins while the burner extinguishes when the water reaches the configured temperature T_2, $T_2 \geq T_1$. The flow action continues until desired air temperature is reached, while the burner may repeat the ignition/extinguishing phase in order to keep the

water temperature close to T_2.

The third and last approach for DES is the *process-based* one. In that case, the model is a collection of processes that represent an entity throughout its life cycle as a sequence of actions and reactions driven by logically related activities. An example of such model is the building's battery storage for which a collection of three activities can be envisioned as a model. Besides the idling activity where no energy is exchanged, a discharge activity and a charge activity model the power flow while enforcing the battery's energy capacity limits. The *process-based* paradigm of DES simulation is easier to comprehend since it resembles real-world objects.

According to Perumalla et al. [210] a pure process-oriented paradigm is following all the features, F1 to F5, listed below. The features F1 - F3 denote that a programming style can be created using conventional programming languages, whereas the F4 and F5 which are regulating the time, require special software architecture design and provisioning.

- F1: procedures can declare and use local variables.
- F2: procedure calls can be nested.
- F3: procedures can be recursive and re-entrant.
- F4: primitives for advancing simulation time can be invoked in any procedure.
- F5: primitives for advancing simulation time can be invoked wherever a conditional, looping or other statements can appear.

Emulation engine design concept

This research work proposes an innovative solution by introducing a hybrid DES paradigm, based on the three design approaches mentioned above, as illustrated in Fig. 4.2.

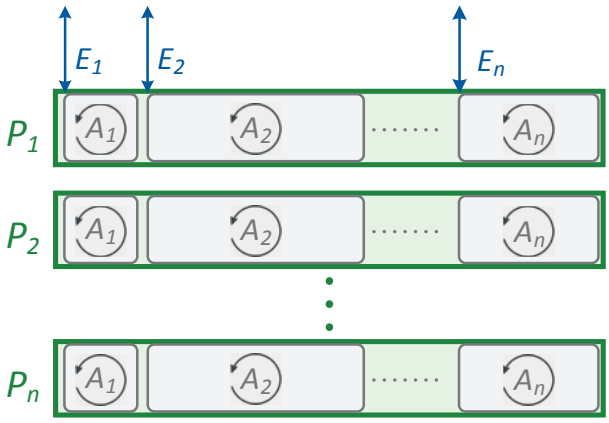


Figure 4.2 – The proposed hybrid DES engine. P_x denotes the processes, A_x the activities and E_x the events

To begin with, the whole building is simulated as a collection of process-based models,

each one representing a real world object, for example, an energy-related infrastructure, a sensor/actuator or an occupant. They execute concurrently, and they support all the features mentioned by Perumalla et al. in [210]. Those individual processes are named virtual entities (*vEntities*). Internally, each vEntity implements a dual, activity and event-driven approach depending on the capabilities and configuration of their model. Therefore, the state of each vEntity (process-based) changes based on the model (activity-based) or external and internal triggers (event-based).

Fig. 4.3 presents the logic of the process, activity, and events for some representative virtualized building components. The vEntities are studied in great detail in the subsection 4.5.3.

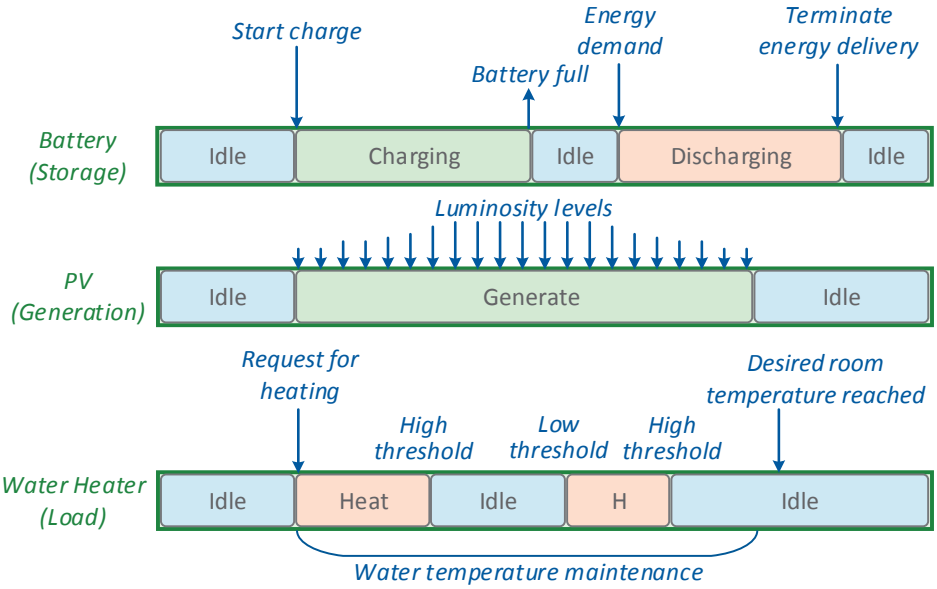


Figure 4.3 – Representative examples of the virtualized infrastructure using the hybrid modeling approach

Concluding, even though the building simulation might seem to be designed as an agent-based simulation (ABS), the frontier with DES is actually thin [211]. In ABS, an agent typically holds intelligence such as rule-based decisions or local optimization algorithms. Its aim is to maximize the benefits for him or its team in case of collaborating agents. Therefore, the overall evolution of the system depends on those agents interaction. On the contrary, vEntities are configured dynamic models, characterized by their activities and various states, while their interactions are solely in the form of events.

4.4.3 Lightweight multithreading mechanism

While literature proposes many solutions for emulating buildings, few if any tackle the aspect of computational efficiency. The novelty of the proposed solution is the BMS integrated and optimized emulation engine. This creates an efficient virtualization system for continuous

operation.

For improving the performance, instead of operating system (OS) processes handling the DES models, this work utilizes the *coroutines*. Those can meet the requirements, F1 - F5, in a much lighter and efficient way.

As micro-thread (uTread) is named the software implementation of such coroutines; it resembles, in fact, a tiny version of a thread. Unlike threads though, the uTreads share most of their memory stack and thus can run with minuscule inherent resource requirements. That enables hundreds of non-CPU bound coroutines to execute in single, common kernel thread. The context switching between uTreads is done explicitly and cooperatively; the uTread's scheduler passes the control to the next one in a round-robin manner, without any OS kernel involvement. The latter together with the high proportion of shared memory stacks enable an ultra-fast context switching, requiring far fewer resources compared to other parallelism architectures [212]. Fig. 4.4 illustrates the difference between an OS process, thread, and uTread.

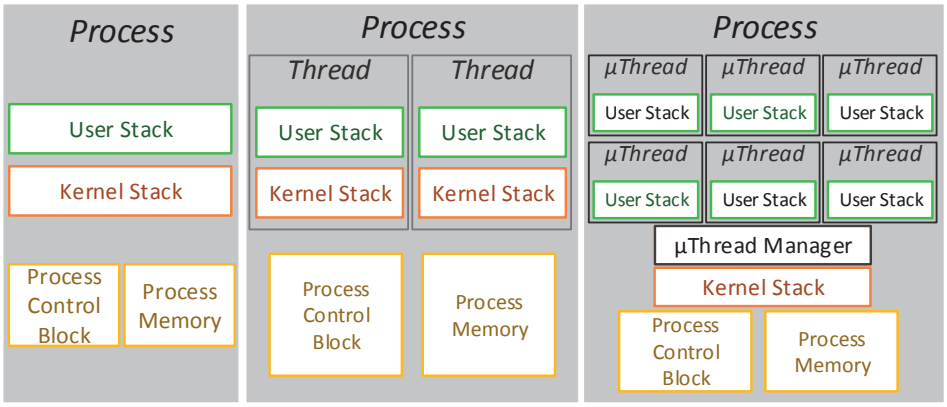


Figure 4.4 – From left to right: regular process, threaded process, micro-threaded process

The pseudo-concurrency offered by those coroutines is not a new notion in the computer science. The micro-threading architecture is extensively used in web-servers and cloud applications due to their I/O bound operation. This micro-threading design is known to outperform the conventional architectures in high concurrency workloads [213, 214].

The similarity in the program flow of process-based DES models and the loads encountered by the web servers, inspired the preliminary testing of uTreads for DES purposes. Similar to the I/O bound operations of web-servers, the DES processes are governed by their activities which most of their time remain in the state until an internal or external event triggers a change. The initial testing and the validation to be found in Section 4.6 justified the choice, enabling hundreds of DES processes to execute, communicate and interact concurrently.

An alternative to uTreads would have been the asynchronous software design, using an event loop and event handlers. Although this approach can achieve the same objectives in addressing

the hybrid DES requirements, it breaks the logical activity flow of the vEntities. The model development becomes more complex to manage, and the modularity of the engine is greatly impacted. Finally, managing local and shared variables of the vEntities requires additional software routines and data structures.

Concluding, the coroutines, for some workloads, show significant advantage compared to regular processes, threads as well as asynchronous designs. In fact, this makes them suitable for actualizing the current theoretical, hybrid DES engine design.

4.5 Emulation Engine Architecture, Implementation, and Operation

This section studies the architecture and implementation of the building emulation engine after being theoretically defended in the previous section. The section is organized in four individual subsections. Firstly, in subsection 4.5.2 the architecture of the building emulator is presented with its core components and their interconnections. Secondly, subsection 4.5.3scrutinizes the modeling and algorithmic design of the solution. Thirdly, subsection 4.5.4 introduces the performance regulation module for enabling the real-time operation. The section concludes with the subsection 4.5.5 that presents a unique, real-time embedded network emulation module.

4.5.1 vMid: the emulation engine as a module of the BMS

The BMS named OpenBMS, cf. Chapter 2, supporting the emulation engine of this work, consists of an end-to-end, layered and event-driven architecture. The key advantage of the particular BMS architecture is the distributed middleware.

The term middleware, cf. Chapter 3, refers to a dynamic pool of low power electronics, distributed in the building for exchanging information crucial for its management. They are essentially acting as intelligent agents of the SB. They not only enforce the optimization functionality but also interconnect the various technologies and protocols found in the current generation of SB. The middleware acts as a delegate and abstraction layer of the underlying control and measurement devices (sensors, actuators), for the management and intelligence services.

The part of the middleware integrating the physical devices and protocols is rightly called physical middleware (pMid). The emulation engine is forming a type of virtual middleware similarly to the pMid. This subset of middleware is called virtualization middleware (vMid).

Both physical and virtual subsets of middleware share the same data model of the messages, as well as communication patterns, sockets, and libraries. Therefore, the pMid and vMid components are completely interoperable between each other and abstracted from upper

layers of building management and services without any additional adaptation layer.

Fig. 4.5 highlights the connectivity scheme for vMid, pMid, and BMS. The latter interfaces both types of middleware through the same zeroMQ module, remaining unaware of the exact nature of the devices. Similarly, the real time server (rtServer) handles both types of incoming events without modifications. Finally, the energy management system and the ambient intelligence can evaluate advanced algorithms by incorporating physical and nonexisting, virtual devices in the same data structures.

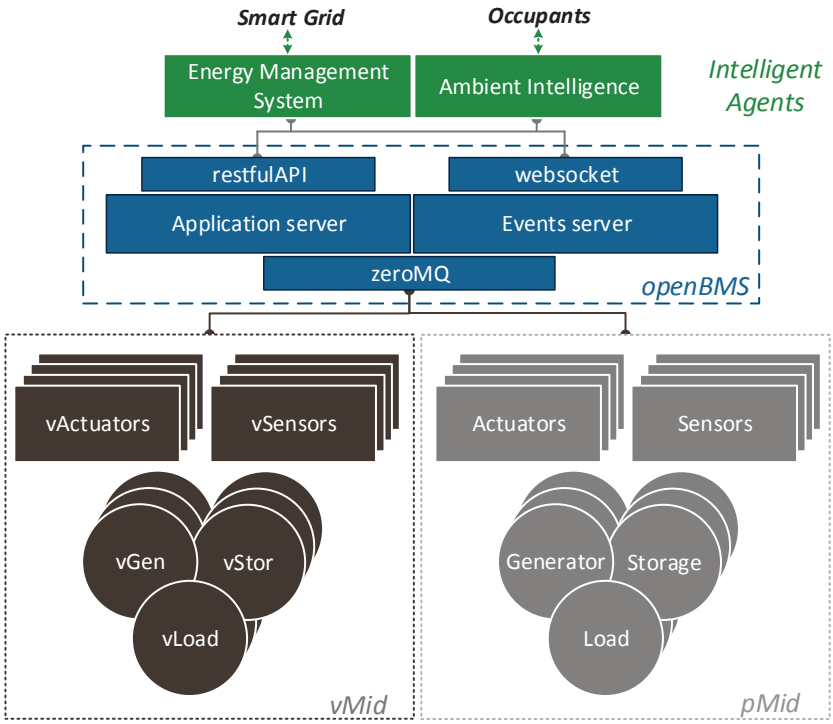


Figure 4.5 – The vMid, pMid, and BMS connectivity scheme

4.5.2 vEngine: the virtual middleware core

vEngine architecture

Virtualization engine (vEngine) refers to the core component of the vMid architecture and it is responsible for implementing the emulation of the various vEntities. Fig. 4.6 depicts its architecture. It consists mainly of three discrete modules, the *vMiddleware manager*, the *vNetwork emulator* and the *vEntities pool*.

Each of those modules is implemented in an individual process for isolation. This offers better parallelization in multi-core architectures and improved scalability using network-distributed processes.

The communication between the three modules and internally in the *vEntities pool* is achieved

using the ZeroMQ sockets and patterns. The software library is the same to the one used by middleware for interfacing the BMS. The high-performance ZeroMQ library is an ideal candidate even for high throughput, inter-process communication.

For implementation purposes, Python was used as a high-level, general-purpose programming language. According to the preliminary testing, it does not introduce any significant overhead to the emulator. This is justified by the fact that most of the core libraries have been written in C++ while the event-driven models are not CPU-bound. Indeed, several high-performance web servers share the same observations, with Python as their programming language of choice.

On the other hand, Python's straightforward and expressive syntax and dynamic typing minimize the implementation time for the emulated models. In addition, its scalability, while remaining scientific oriented, makes it a unique fit for such system targeting a scientifically proven, yet product-oriented emulation tool.

This subsection introduces the architecture of the vEngine and scrutinizes the *vMiddleware manager*. The *vNetwork emulator* and the *vEntities pool* are detailed in the separate subsections 4.5.5 and 4.5.3 respectively.

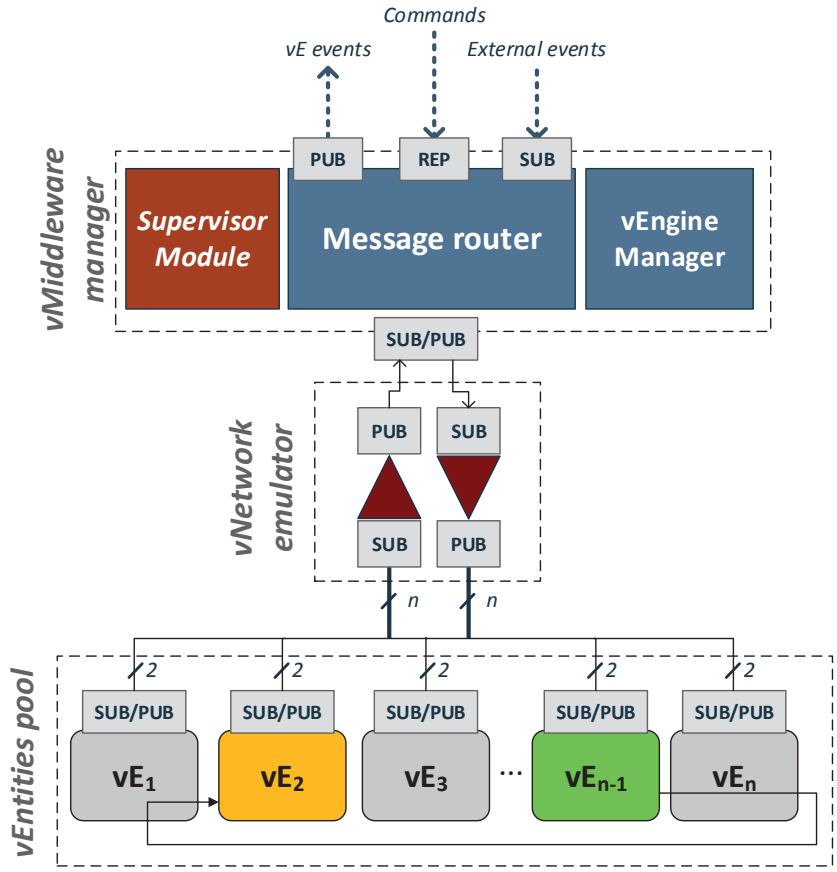


Figure 4.6 – The vEngine architecture: (green) a vE executing, (yellow) vE waiting for the program control and (grey) vE cooperatively deferred execution

vMiddleware manager

The *vMiddleware manager* is the primary module of the engine and handles both the communication and coordination aspects of the engine. It is thanks to its adaptation layer that the vEngine participates seemingly into the middleware concept. The compatibility with the middleware requires specific data structures and ZeroMQ sockets which are both handled by the message router visible in Fig. 4.6.

The message router features various types of communication sockets depending on the type and scope of the exchanged messages:

- a socket of PUB type for publishing events to the middleware;
- a socket of SUB type for receiving external events;
- a socket of REP type for receiving synchronous commands from the BMS;
- a SUB/PUB socket pair for inter-process communication with the vEntities pool.

While the vMid maintains external compatibility with the pMid, the internal data model of the messages between the *vMiddleware manager* and the vEntities pool is customized to the needs of emulator. Internally, a type of multiple-frame packet format is used, called multi-part message. Its 4 parts are specified as follows:

1. a message recipient, as vEntity ID or *vMiddleware manager* gateway ID;
2. a message sender, as vEntity ID or *vMiddleware manager* gateway ID;
3. a UNIX timestamp;
4. an actual payload encoded in JSON notation, similar to the middleware messages' payload.

The multi-part messages have several performance advantages for high data-volume designs like the emulator.

- It keeps the information required for routing such as the sender and receiver, in a separate frame. Hence, deep copy and parsing of the packet is avoided during the various routing stages. This allows a zero-copy approach for high-volumetric communication simply by passing the payload as a reference to a memory location. On the contrary, in monolithic messages, the body must be copied into a temporary memory buffer and parsed in order to extract the data required for the routing.
- The message rerouting leaves the original payload untouched, avoiding expensive de-/serialization operations.
- The routing algorithms are simplified and agnostic to the payload since all the addressing information are found in a separate part of the message.
- It enhances the expandability of the messaging data models since additional parts can be chained on the multi-part message by intermediate brokers and routers without

violating the primary protocol.

- Finally, while from the application point of view, the message parts are logically independent; on the transport network layer, they are transferred as a single entity. This guarantees the delivery either of all or none of the parts, avoiding synchronization issues.

The coordination aspect of *vMiddleware manager* is handled by the vEngine Manager visible in Fig. 4.6. Its tasks include: the instantiation of vEntities pool according to the models configured in the BMS database, the configuration of the *vNetwork emulator* and finally, the startup, shutdown and state backup of each vEntity. Finally, the *vMiddleware manager* is also the execution entry point of the engine and the instance which forks the threads and processes for each module of the vMid.

Each single vEntity gets its model and simulation parameters upon thread spawning. One of them, the relative time expansion parameter $k_{dt} \geq 1$ is defining a relative time space for the simulated activity. When $k_{dt} = 1$, the virtualization executes in real time with its physical counterparts, thus emulating the virtual components. Whereas, when $k_{dt} > 1$, the emulation is accelerated relatively to the pMid. This simulation mode utilizes the vEngine independently from the CPS of the SB.

Fig. 4.7 illustrates the interactions of the *vEngine* with the rest of the BMS in the form of a sequential unified modeling language (UML) diagram. The diagram illustrates the startup and continuous operation sequence which are described below.

The initialization sequence is colored with red in the UML diagram. During startup, the *vMiddleware manager* performs an application programming interface (API) request to the building managing server. The reply includes the parameters of each vEntity that are configured on the server. Additionally, it includes all the information crucial for the vMid operations, such as its IP:port combinations, socket configurations as well as the IP locations of the rest of the pMid modules. The latter enables the distributed communication and self-discovery features of the middleware. Hence, only the location of the management server is statically defined; the middleware topology is fetched and dynamically created and modified during runtime.

A major part of the UML diagram is colored with green and illustrates the continued operation of the engine. The "par vEntity" denotes the parallel executing uTreads of the vEntities pool. The "loop", on the other hand, denotes the continuous *vMiddleware manager* operation. Hence, it handles the commands processing, the events forwarding as well as the performance regulation.

Furthermore, with blue is colored the external, and physical events originating from the pMid and the users. There are two types of events. Firstly, the user and API ones pass through the application server which converts them to synchronous action requests to the *vMiddleware manager*. Secondly, there are also the asynchronous events originating from the sensor networks and other agents of the pMid. For those type of events, a dedicated thread monitors

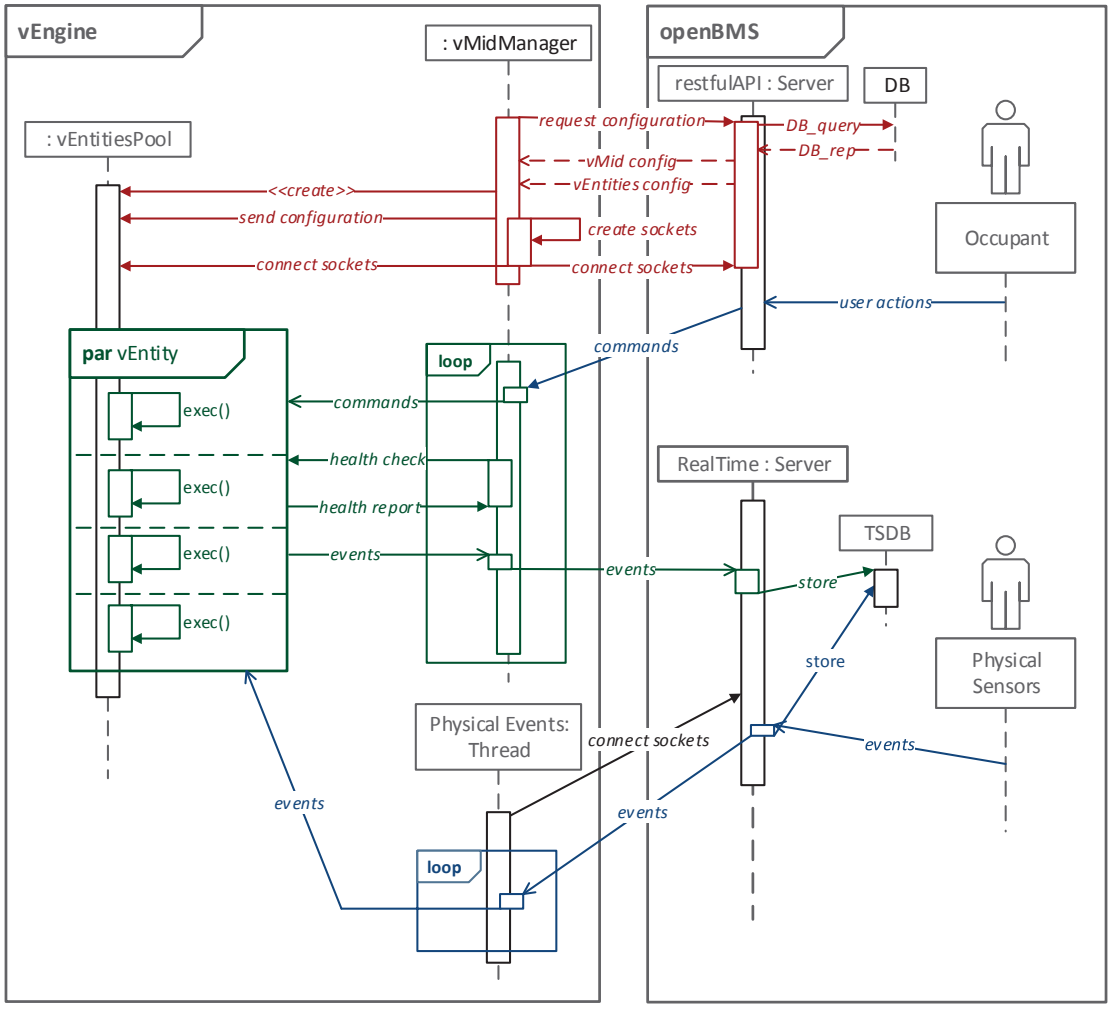


Figure 4.7 – The sequential UML diagram of the vEngine interactions with the rest of the BMS

the appropriate sockets and receives the external events. It then converts them to the data model used by the vEntities pool before pushing them to it.

4.5.3 vEntities: the core of emulation

vEntity architecture

The fundamental elements of the vEngine are the virtual components called vEntities. They are implementing the component models for creating the emulation. Practically implemented as uTreads, they form a "pool" connected to the *vMiddleware manager* through the *vNetwork* for the events exchange purposes. The vEntities are both generators and receivers of events. Thus, they interact with each other for commands and events exchange.

Due to the disparate types of entities, a standardization of their process, activities, events, and

configuration is required. By doing so, a *universal* initialization, execution, and algorithmic model design approach can be followed. Moreover, this universal structure is compatible with of the DES framework according to the formalism introduced in Section 4.4. Fig. 4.8(a) displays the flowchart and Fig. 4.8(b) presents the sequential UML diagram that each vEntity features.

Each block in the flowchart represents a specific activity of the vEntity:

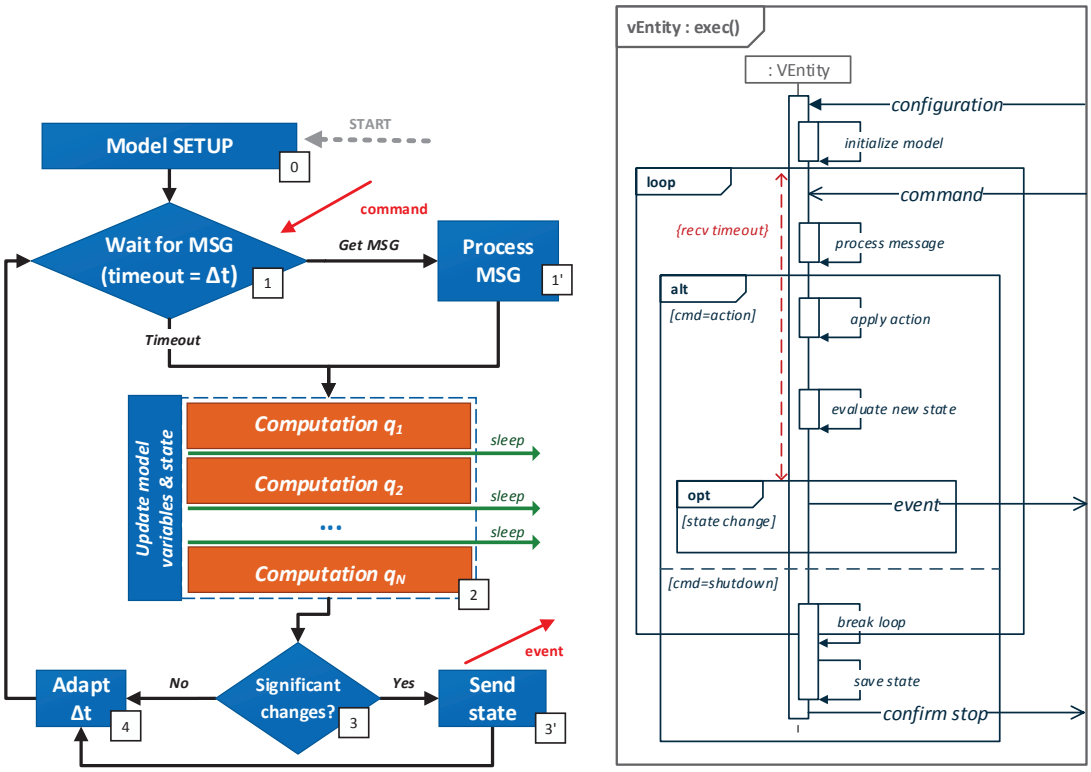
- **(0)** Set up the parameters, variables and states of the entity's model. This step gives a unique identity to the vEntity.
- **(1)** Sleep for Δt seconds while waiting for external events.
- **(1')** If an event from the *vMiddleware manager* has triggered the awakening, process this message by updating specific variables.
- **(2)** Computations based on the model aim to update internal variables and states due to events or model activities.
- **(3)** Based on specified thresholds, the vEntity assesses whether its internal state has considerably changed.
- **(3')** In case of a significant change, the vEntity sends the corresponding event to the virtual network (vNetwork).
- **(4)** The vEntity self-adapts its suspend time according to an estimation until the next event triggers or the internal activity changes.

The sequential UML diagram, on the other hand, illustrates in a self-explanatory way the functionality of each vEntity, throughout its life-cycle. The "**loop**" denotes the continuous and it is equivalent to (1) to (4) flowchart blocks. During that time, the uTread remains suspended, unless an event triggers it, for "recv timeout"=Δt seconds. Additionally, the functionality in "**opt**" is activated only if there is a state change. Finally, the "**alt**" triggers the shutdown and state saving of each vEntity when requested by the manager. The latter eventually enables a "snapshot" of the vEngine state to be taken. Hence, restarting, scaling up, and even transferring to different hardware can be executed without downtime.

The aforementioned flowchart forms a common wrapper for each vEntity that facilitates their intercommunication and their monitoring from the upper layer manager. The latter takes advantage of the ability of any entity to dynamically adapt its suspend time and hence, free up engine's resources.

Moreover, as the computations step (Q), cf. Fig. 4.8(a), might be computationally heavy, it should be designed in a way that the running uTreads cannot consecutively use the CPU more than a given quantum time q_x where

$$Q = \sum_{x=1}^{N} q_x \tag{4.3}$$



(a) Flowchart

(b) Sequential UML diagram

Figure 4.8 – vEntity implementation flowchart and UML diagrams: (0) initialization, (1) suspended while waiting for an event/timeout, (2) update state, (3) output event, and (4) adapt suspend time

In between these quantum execution blocks, the program control is returned; the next vEntity can now acquire it. That design technique enables a *cooperative* parallelism without limiting the complexity of model's algorithm.

As soon as a vEntity is instantiated, it retrieves both the simulation parameters and the simulation model from the BMS. On the one hand, the simulation configuration holds the list of values to parameterize the aforementioned steps from (1) to (4), regardless of the entity type. On the other hand, the simulation model links the variables to the states and goes through the specific software-class logic. As the vEntity itself can control its wake up frequency, continuous time models may run along with the DES ones, leading to a *hybrid* simulation.

Listing 3 shows the typical structure of the simulation parameters transferred to each vEntity as part of the initial configuration. The most important element is the `ve_class`. It indicates to the *vMiddleware manager* which algorithm logic has to be executed for that particular model.

The parameter `dt_timeout` defines the Δt timeout used to control the awakening frequency in (1). The user has the freedom to use a statistical probability density function (PDF) to generate it, although a fixed value may also be used. The PDF one is mainly to mimic the event-based

nature of many sensors which send a measurement asynchronously instead of sampling over time intervals. The field `variables` lists the interface's variables (input/output), whereas the `events_to_var_in` and `var_out_thres` bridge the vEntity to the rest of the pool. The former indicates to the *vMiddleware manager* how to match the input variables with the events produced by other vEntities, or any external source (physical sensors, static file, etc). Referring to the RTDEVS formalism, these fields define the sets X, Y, Z of the RT AM and RT CM models, directly or indirectly. The `var_out_thres` refers to the output variable change threshold for the vEntity to trigger an event, cf. Fig. 4.8(a). Detailed in Section 4.5.4, the last optional field `thres_to_dt` characterizes the influence threshold of an output change over the suspend time Δt.

```
{
   "ve_class": <class_name>,
   "dt_timeout": [<pdf_type>, <pdf_param>],
   "variables": {
      "in": [<var_in_1>,...,<var_in_n>],
      "out": [<var_out_1>,...,<var_out_n>]
   },
   "events_to_var_in": {
      "var_in_1": [<source_in_1>,<type_in_1>],
      ...,
      "var_in_n": [<source_in_n>,<type_in_n>],
   },
   "var_out_thres": {
      "var_out_1": <delta_out_1>,
      ...,
      "var_out_n": <delta_out_n>,
   },
   "thres_to_dt": {
      "delta_out_1": <dt_1>,
      ...,
      "delta_out_n": <dt_n>,
   }
}
```

Listing 3 – Structure of the simulation parameters transferred from the BMS to the vEntities

Virtual `root` model

The model class given by `ve_class` implements the steps (0, 2, 4) of the described flowchart. Referring to the DEVS formalism, the setup phase (0) defines the internal and external transition functions δ_{int} and δ_{ext}. Whenever possible, the model designer might as well define how the suspend time Δt should be adapted based on internal or external variables, in order to reduce unnecessary awakenings. As many building components, and thus models, share simulation features, the notion of parent classes has been created. Child classes are implementing a particular feature, are inheriting them in order to customize and extend them, without the need to redefine the standardized features, including the vEntity's communication frontend.

The following subsections describe some implemented vEntity models for validating the building virtualization solution. In future work, or with the help of the open-source community,

the parent classes can be used for designing more elaborate and accurate representation of building components. The scope of this work was primarily to present the innovative architecture of the engine and highlight its features, rather than to promote simulation models that are ahead of state of the art.

Virtual ambient sensors and actuators: `vSensor` **&** `vActuator`

The future SB essentially relies on a broad network of sensors and actuators, enabling data analytics. The ambient sensors mainly collect room temperature, humidity, and luminosity, along with human presence. Concerning the actuators, various functions may be envisioned, such as automatic window blinds, doors and locks, light dimmers, and any local controllers.

To begin with, their virtualization model doesn't differ much from the generic vEntity parent-class. Their primary functionality consists in interconnecting and reacting on outputs of other physical or virtual entities. For example, a virtual motion sensor may be configured to trigger when the luminosity (physical sensor) of the room abruptly increases. Similarly, a virtual dimmer can trigger a virtual load when a physical motion sensor detects movement in the room. Furthermore, besides the reactions to external stimulus, the virtual ambient sensors and actuators can operate based predefined values (e.g. static values file).

Hence, the *vSensor* and *vActuator* classes offer the possibility to linearly associate output variables with input events or static values, cf. Listing 3. The Algorithm 3 describes the procedure for the *vSensor*.

Algorithm 3 vSensor parent-class logic

1: **procedure** SETUP_MODEL($vSensor_{conf}$)
2: $c \leftarrow vSensor.coefficients$
3: **end procedure**
4: **procedure** PROCESS_MSG(msg) ▷ message from other vEntity
5: $v_i \leftarrow msg$["payload"] ▷ match to an input variable
6: **end procedure**
7: **procedure** UPDATE_STATES
8: **for all** $v_o \in var\_out\_list, v_i \in var\_in\_list$ **do**
9: $v_o \leftarrow \sum c(v_o, v_i) * v_i$
10: SEND_MSG($vmid_{addr}, v_o$) ▷ forward upwards
11: **end for**
12: **end procedure**

The separate *vSensor* and *vActuator* entities instead of an integrated functionality into the root model has an important advantage. The discrete cyber (*vSensor, vActuator*) and physical (*vLoad, vStorage, vGeneration*) entities combination is a more realistic representation of the architectures used in the SB. Thus, the models and functionality that describe each type of infrastructure is better isolated.

For example, the emulated powers and states of a *vLoad* are internally transferred to a *vSensor*, which models the process of sampling and transmission of the data. The thresholds for event creation, the possible systematic or random errors in measurements, the digital networks particularities and others, are only part of the *vSensor* model.

Listing 4 reveals the vEntity configuration parameters for controllable window blinds which influence the room luminosity. Firstly, the configuration defines the actual entity class to be used. The internal variables of interest are the requested blinds angle and the outside irradiance. The output value is the calculated luminance which is then transferred to a luminosity *vSensor*. Besides the generic simulation parameters, "`sim_param`", it includes additionally the "`model_param`" ones. As the name suggests, the latter configures the exact type of blinds output luminosity, as a weighted influence of their angle and the outside irradiance.

```
{
   "sim_param": {
      "ve_class": "vActuator",
      "variables": {
         "in": ["angle", "irr"], "out": ["lum"]
      },
      "events_to_var_in": {
         "angle": ["vUser", "command"]
      },
      "var_out_thres": { "lum": 10 }
   },
   "model_param": {
      "weights": {
         "lum": { "angle": 1.11, "irr": 1}
      }
   }
}
```

Listing 4 – vBlind simulation and model parameters

Virtual energy storage systems: `vStorage`

The storage system model all the infrastructure able to store any form of energy, which they release it afterward. They play a significant role in facilitating the energy management strategies. A characteristic *vStorage* entity is the notion of state of charge (SoC). It defines the percentage of the total energy capacity that is currently available in the virtual component. Due to storage system particularities, the SoC varies based on the requested power P. Moreover, the available power P depends on the current SoC. These make the storage simulations a challenging task. Finally, depending the combination of SoC and the power flow direction the storage system can be in four states: ["charge", "discharge", "empty", "full"].

Algorithm 4 summarizes the logic *vStorage* class. The $updateSOC()$ function definition is leaving the implementation to the sub-class designer. The ΔSOC^{thres} on the other hand governs the output events frequency.

While any form of energy can be represented with the *vStorage* sub-classes, the lithium-ion

battery is a excellent solution for residential energy storage. Its update of the SoC over Δt according to [215] is

$$SOC(t+\Delta t) \;=\; SOC(t) + \eta \cdot P \cdot \frac{\Delta t}{C} \tag{4.4}$$

where η is the electrical-to-chemical efficiency, C the total capacity, and P the power flow. The parameter η depends on both SoC and P while varying over time; its characterization thought is beyond the scope of this work.

Algorithm 4 vStorage parent-class logic

```
1: procedure SETUP_MODEL(vStorage_conf, saved_state)
2:     if saved_state then
3:         SOC ← saved_state.soc
4:     else
5:         SOC ← 0
6:     end if
7:     P_actual ← 0
8: end procedure
9: procedure UPDATE_STATES(P_req)                    ▷ requested ± power
10:    if (SOC = 0 and P_req < 0) or (SOC = 100 and P_req > 0) then
11:        P_actual ← 0
12:    end if
13:    SOC ← updateSOC(P_actual, SOC)
14: end procedure
15: procedure ADAPT_DT
16:    dt ← C / P_actual * ΔSOC^thres
17: end procedure
```

Virtual energy production: `vGeneration`

A virtual generation system *vGeneration* has been designed for modeling any sort of electricity production in the building, most commonly in the form of renewable resources. Very similar to the implementation of *vSensor*, it gathers instantaneous environmental data for emulating the output power.

In order to illustrate type of vEntity, the sub-class *vPVpanel* is presented. It emulates the events of a power sensor connected to a photovoltaic system. More specifically, it is composed of a photovoltaics (PV) array whose power depend on the current irradiance $G(t)$ the cell temperature $T(t)$. The power curves and maximum power point can be computed from the one-diode model [216]. However, this requires demanding mathematical operations which do not fit in this cooperative, real-time engine.

Hence, the linearization [217], as seen in Eq. 4.5, of the output power of the PV panel has been

used instead .

$$
\begin{aligned}
P_{max}\big(G(t),T(t)\big) &= \frac{G(t)}{G_{STC}} \cdot \Big(P_{STC} + P_{max}\big(T(t)\big)\Big) - k_{loss} \cdot P_{STC} \\
&= \frac{G(t)}{G_{STC}} \cdot \Big(P_{STC} + P_{STC} \cdot C_p \cdot (T(t) - T_{STC})\Big) - k_{loss} \cdot P_{STC} \\
&= P_{STC} \cdot \Big(\frac{G(t)}{G_{STC}} \cdot \big(1 + C_p \cdot (T(t) - T_{STC})\big) - k_{loss}\Big)
\end{aligned}
\tag{4.5}
$$

where P_{STC} is the power at standard test conditions (STC), G_{STC} is the irradiance at STC, C_p is the power temperature coefficient, T_{STC} is the reference STC temperature, and k_{loss} represents the losses coefficient. The PV cell manufacturer provides those data, and with those, a linearized value of P_{max} can be obtained. The STC correspond to irradiance at $1000 W/m^2$, cell temperature at 25 °C, and air mass coefficient of l.5.

Fig. 4.9 validates the proposed formula by plotting the maximum power relative error between the *one-diode* model simulation, and its linear simplification. For solar irradiance above 200 W/m^2, the model linearization yields sufficient precision ($\leq$ 10% relative error). Under 200 W/m^2, the error increases until eventually reaching 100%. Given the low energy production at these irradiance levels, the absolute error remains in the order of few $Watt$ per cell.

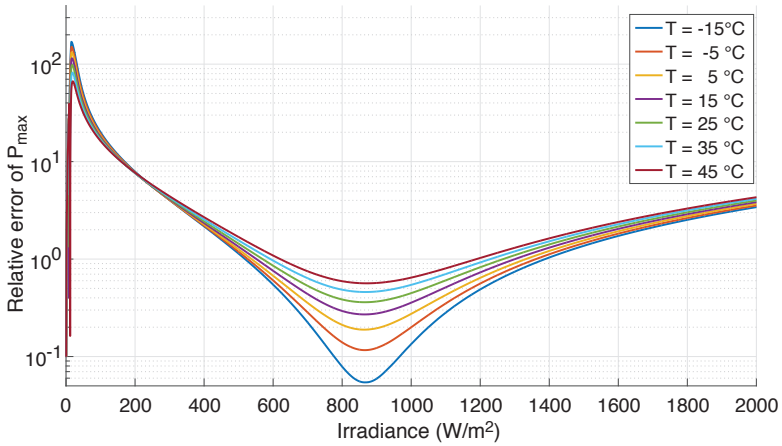


Figure 4.9 – Maximum power relative error between the one-diode model simulation and its linear simplification

Listing 5 shows the simulation and model parameters of the MaxPower CS6X-310P PV panel. The nominal P_{STC} is 310 W, the power temperature coefficient $-0.43\%/°C$ and the losses 1%.

```
{
   "sim_param": {
      "ve_class": "vPVpanel",
      "variables": {
         "in": ["irr","temp"], "out": ["P"]
      },
      "events_to_var_in": {
         "irr": ["vOutSensor", "irr"],
         "temp": ["vOutSensor", "temp"]
      },
      "var_out_thres": { "P": 1 }
   },
   "model_param": {
      "P_stc": 310, "T_stc": 25, "G_stc": 1000,
      "C_p": -0.0043, "k_loss": 0.01
   }
}
```

Listing 5 – vPVpanel simulation and model parameters

Virtual energy consumers: `vLoad`

Any device that consumes power is part of the *vload* category; for instance, a lighting system, a computer or a heat pump. Each device of this type inherits the *vLoad* class that handles the events generation and virtual power consumption. The characterization of that power profile over time is achieved with the notion of load profile (LP), a structure composed of three sets:

1. **Modes** ($mode$): a list of load power ranges, linked to the various operational states of the load e.g. normal mode, sleep mode, washing cycle, etc.
2. **Sequences** (seq): a set of structures that statistically describes the mode power values and its transitions.
3. **Activities** (act): an ordered list that describes the deterministic or externally triggered state changes of the load.

The *vLoad* uses the above structures to emulate the virtual power. A part of the *vLoad*'s logic, seen in Algorithm 5, describes the process of emulation for the above sets 1 and 2. Nevertheless, the emulated power is finally transferred to a *vSensor* type of entity for implementing the power events.

Any sub-classes of *vLoad*, for instance a virtual lamp, a heat pump, a heater, etc., leverage and fine-tune this LP parameters. Hence, the *vLoad* entity can emulate any type of device from the power use point of view.

Listing 6 reveals the vEntity configuration parameters for a virtual computer, *vComputer*. The listing indicates that *vComputer* is triggering state events following a normal distribution $\mathcal{N}(\mu, \sigma^2) = \mathcal{N}(10,5)$, while generating a virtual power "P" with an accuracy of 1 $Watt$ and expecting commands from a virtual user $vUser$. The model parameters configure two modes of operation [70 ; 150] $Watt$ and [180 ; 240] $Watt$. The first one lasts for minimum $\mathcal{N}\{20,10\}$

minutes with a probability of mode change equal to 0.15. Similarly, the second one lasts for minimum $\mathcal{N}\{10,2\}$ minutes with a probability of returning to first mode equal to 0.85.

Algorithm 5 vLoad parent-class logic

```
1:  procedure SETUP_MODEL(vLoad_conf, saved_state)
2:      seq_list ← vLoad.sequences
3:      mode_list ← vLoad.modes
4:      if saved_state then
5:          load.state ← saved_state                    ▷ restore load's state
6:          POWER_VALUES(load.state)
7:      else
8:          POWER_VALUES(vLoad.initial_state)
9:      end if
10: end procedure
11: procedure PROCESS_MSG(msg)                  ▷ message from load's actuator
12:     load.state ← msg["command"]
13:     POWER_VALUES(load.state)
14: end procedure
15: procedure POWER_VALUES(state)                ▷ power values of load state
16:     mode ← retrieve_mode(mode_list, state)
17:     seq ← retrieve_sequence(seq_list, mode)
18:     power_values ← generate_power_values(seq)
19:     power_index ← 0                          ▷ indexes current power value
20: end procedure
21: procedure UPDATE_STATES
22:     if power_index < length(power_values) then
23:         P ← power_values(power_index)
24:         addr ← load.sensor                      ▷ get connected sensor
25:         SEND_MSG(addr, P)
26:     else                                        ▷ get new random powers
27:         POWER_VALUES(load.state)
28:     end if
29:     power_index ← power_index + 1
30: end procedure
```

```
{
   "sim_param": {
      "ve_class": "vLoad",
      "dt_timeout": ["NORM", [10, 5]],
      "variables": {
         "in": ["act"], "out": ["P"]
      },
      "events_to_var_in": {
         "act": ["vUser", "command"]
      },
      "var_out_thres": { "P": 1 }
   },
   "model_param": {
      "mode": {
         "low": [70, 150], "high": [180, 240]
      },
      "seq": {
         "default" : "low",
         "low": {"dur": [20, 10], "high": 0.15},
         "high": {"dur": [10, 2], "low": 0.85}
      },
      "act": [[8, 4], [13.5, 4.5]]
   }
}
```

Listing 6 – vComputer simulation and model parameters

Virtual occupant activities: `vUser`

The *vUser* is the final major type of vEntity. It enables the emulation of virtual occupant behavior by acting on the *vActuators*. The logic governing this entity is described in Algorithm 6. A *vUser* retrieves from the BMS the LP activities, act (3), of each of the virtual loads it controls. It subsequently triggers the appropriate *vActuators* for each individual virtual load based on the the activities set.

The current model of *vUser* is limited to predefined actions and scenarios. Nonetheless, dynamic models could extend it, taking into account several external signals such as actual human presence in the building, historical data on their activities and other. This way, the advanced behavioral studies, mentioned at the beginning of the chapter, can be implemented.

Algorithm 6 vUser parent-class logic

```
1: procedure SETUP_MODEL(vUser_conf)
2:     vLoads ← vUser.loads
3:     for all vLoad ∈ vLoads do
4:         act_list ← act_list.append(vLoad_act)        ▷ user activities on all loads
5:     end for
6: end procedure
7: procedure UPDATE_STATES
8:     for all act ∈ act_list do
9:         t_n ← time_now()
10:        if t_n ≥ act.start then                      ▷ start time passed
11:            ACTIVATE(act.actions)                    ▷ apply actions of activity
12:        end if
13:        if t_n ≥ act.end then                        ▷ stop time passed
14:            DEACTIVATE(act.actions)                  ▷ restore actions of activity
15:        end if
16:    end for
17: end procedure
18: procedure ACTIVATE(actions_list)
19:    for all action ∈ actions_list do
20:        action.run ← True
21:        load ← action.load
22:        addr ← load.actuator                         ▷ get connected actuator
23:        cmd ← action.cmd                             ▷ specific actuation command
24:        SEND_MSG(addr, cmd)
25:    end for
26: end procedure
27: procedure DEACTIVATE(actions_list)
28:    for all action ∈ actions_list do
29:        action.run ← False
30:        load ← action.load
31:        addr ← load.actuator                         ▷ get connected actuator
32:        cmd ← "restore"                              ▷ restore actuator state
33:        SEND_MSG(addr, cmd)
34:    end for
35: end procedure
36: procedure ADAPT_DT
37:    t_n ← time_now()
38:    dt ← min(next_act.start, next_act.end)
39: end procedure
```

4.5.4 Supervisor: the performance regulator

In order to monitor the real-time operation of the vEngine, a *supervisor* module was created, cf. Fig. 4.6. It acts as a performance regulator, especially for the limited capabilities hardware.

In fact, the cooperative nature of the vEntities execution model necessitate such monitoring module for guaranteeing the computational fairness. The execution model of the vEntities pool is seen in Fig. 4.10. A vEntity can be in one of the states: executing (green), suspended (gray), standby (blue). In this queue-like design, since only one entity can execute at any given time, long computational step (Q) by any of them will impact the whole pool. To make matters worse, unless there are adequate moments of inactivity for the pool to catch-up, the delays will create a cascade effect impairing the operation of the engine.

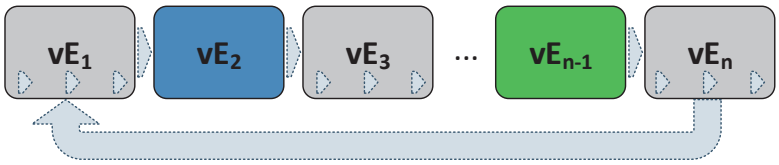


Figure 4.10 – vEntities pool state at any given time. vEntity: green the currently executing, gray the suspended and blue in standby

Ideally, there should is plenty of time between the moment that a vEntity releases the CPU and the moment another one requests it, as the vEntities spend most of their time suspended. However, the cooperative nature does not guarantee the absence of overlapping CPU requests. In fact, the amount of them increases with the number of executing vEntities.

The *supervisor* module is periodically monitoring the vEntities pool for assessing the following features for each entity $vE_i, \forall i \in [1,n]$:

- Q_i: the execution time interval, from entity resume to the CPU execution release;
- Δt_i: the requested suspend time interval;
- Δt_i^a: the actual suspend time interval, from the CPU execution release to subsequent CPU acquiring.
- $\bar{d}_i$ moving average of the experienced delay.

Those quantities are computed by each individual vEntity within a specified time window and collected by the monitoring module. The aforementioned delay is defined as: $(\Delta t_i^a - \Delta t_i) \in [0, d_i^{max}]$. The worst case scenario appears when the wake up events of the vEntities synchronize and thus, all of them wait for the CPU execution. The resulting delay is calculated using Eq. 4.6 for $vE_i, \forall i \in [1,n]$.

$$d_i^{max} = max(\Delta t_i^a - \Delta t_i) = \sum_{j \neq i}^{n} Q_j \qquad \forall i \in [1,n] \tag{4.6}$$

Fig. 4.11 illustrates a hypothetical 4-*vEntities* execution time-frame in which overlapping requests can influence the real time operation. The time-frame starts at *marker 1* with the vE_1 having the control. Before the end of its computational step, the vE_3 timeouts and requests the CPU at $t = \Delta t_3 | \Delta t_3 < Q_1$. However, the CPU control is passed only at $t = \Delta t_3^a$ with a delay of $d_3 = \Delta t_3^a - \Delta t_3$. In the meantime, vE_4 at $t = \Delta t_4 | \Delta t_3 < \Delta t_4 < Q_1$ requests also the CPU, thus waiting in second position. The vE_4 execution eventually begins on *marker 2* and terminates at *marker 3*, having been delayed for $d_4 = \Delta t_4^a - \Delta t_4$. On the positive side, after the vE_4 finishes no other vEntity is in standby, thus the vE_2 can start on *marker 4* with no delay.

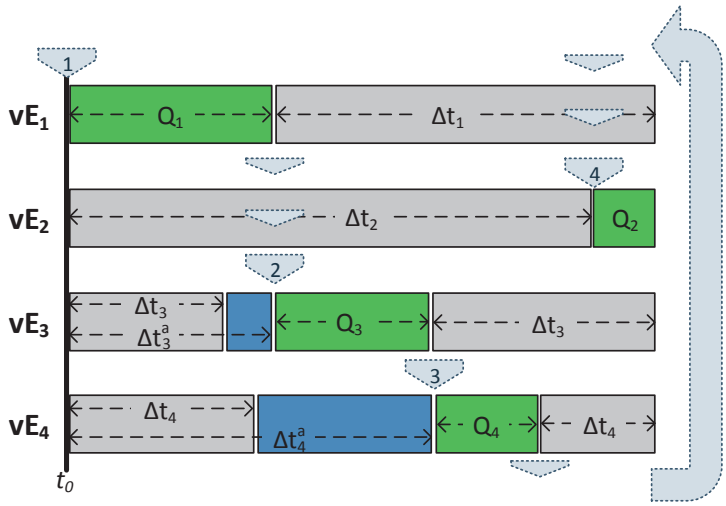


Figure 4.11 – Worst case scenario for 4 vEntities

As d_i^{max} depends on the entity models, their population, and the hosting hardware, the *supervisor* module must dynamically ensure that the delays experienced by any vE_i stay below a certain threshold d_i^{th}. This value is dynamically configured depending on the emulation interests and scenarios. A higher value allows a larger number of vEntities to run even on the weakest of hardware in exchange for relaxed timings. On the other hand, a lower threshold ensures a real-time and highly responsive emulation engine.

Besides the delays monitoring, the *supervisor* acts also as cooperative performance regulator. The chosen solution consists of a punishment algorithm. The *unfair* entities lose priority whenever they might entail excessive delays to their peers. To this end, the per vEntity relative CPU use α_i, mean relative CPU use μ_α, and mean CPU use μ_Q metrics are introduced by Equations 4.7a, 4.7b, and 4.7c accordingly.

$$\alpha_i = \frac{Q_i}{Q_i + \Delta t_i^a} \qquad \forall i \in [1, n] \tag{4.7a}$$

$$\mu_\alpha = \frac{\sum_i^n \alpha_i}{n} \tag{4.7b}$$

$$\mu_Q = \frac{\sum_i^n Q_i}{n} \tag{4.7c}$$

The logic of the engine *supervisor* is deterministic. Instead of directly penalizing the computationally heavy entities, it *dynamically* determines whether the set $\mathcal{U}_{vE}$ of *unfair* entities has an impact on the others due to resources shortage. An entity is determined as *unfair* by any of the two conditions:

- if the vE_i relative CPU use α_i exceeds the mean relative CPU use μ_α by more than σ_α, $(\alpha_i - \mu_\alpha) \geq \sigma_\alpha$;
- if the vE_i CPU use Q_i exceeds the mean CPU use by more than σ_Q, $(Q_i - \mu_Q) \geq \sigma_Q$.

Algorithm 7 illustrates in detail the regulation process. If it exists at least one *unfair* entity, the inflicted delay, $\bar{d}$, to its peers is checked. The impact is considered significant if $\exists\, i : d_i^{max} \geq d_i^{th}$.

To mitigate the impact, either the engine hardware should be upgraded or the offending entity to become more cooperative. The latter can be achieved in three ways. Firstly the vEntity can dynamically adapt its model complexity if it is supported. Secondly, it can distribute in time the computational burden Q into q_x blocks ,cf. Fig. 4.8(a). The vEntity thus releases the CPU execution giving the opportunity to the rest of the vEntities pool to perform their activities. This approach allows some vEntities to implement mathematically complex models without impacting the rest of the pool. As a final resort, vEntity can increase its suspend time interval, the new one is computed by Eq. 4.8.

$$\begin{aligned} \alpha'_i &= \mu_\alpha \Leftrightarrow \\ \frac{Q_i}{Q_i + \Delta t'^a_i} &= \mu_\alpha \Leftrightarrow \\ \Delta t'_i \simeq \Delta t'^a_i &= \frac{Q_i}{\mu_\alpha} - Q_i \quad \forall i \in \mathcal{U}_{vE} \end{aligned} \tag{4.8}$$

Algorithm 7 Health Check module logic

procedure MONITOR_POOL
  $n \leftarrow |vE|$
  $\mathcal{U}_{vE} \leftarrow \{i \mid (\alpha_i - \mu_\alpha) \geq \sigma_\alpha$ **or** $(Q_i - \mu_Q) \geq \sigma_Q\}$
  $\mathcal{F}_{vE} \leftarrow \mathcal{U}^{\complement}_{vE}$
  if $\mathcal{U}_{vE} \neq \emptyset$ **then**
    for all $j \in \mathcal{F}_{vE}$ **do**
      if $\bar{d}_j \geq d_j^{th}$ **then**
        ENFORCE_FAIRNESS($\forall i,\ i \in \mathcal{U}_{vE}$)
        MONITOR_POOL
      end if
    end for
  end if
end procedure
procedure ENFORCE_FAIRNESS(x)
  if vE_x **has a** *simpler model* **then**
    REDUCE_COMPLEXITY(x)
  else if vE_x **has a** *quantized model* **then**
    ENFORCE_QUANTIZATION(x)
  else
    $\Delta t_x \leftarrow \frac{Q_i}{\mu_\alpha} - Q_i$
  end if
end procedure

4.5.5 vNetwork: the embedded network emulator

While the models of subsection 4.5.3 simulate the infrastructure from the behavioral and activity point of view, they ignore the sensor and actuator network characteristics. For example, although a load behavior can be modeled, the feedback of the monitoring and the resulting actions are not performed instantaneously. Most of the time, performance limited, consumer grade, computer and embedded networks are involved in the transmission of the digital packets. The latter leads to additional delays and possible packet losses unforeseen by the load model.

Hence, research and development time has been allocated for also emulating the behavior and uncertainties introduced by the ICT systems. The vNetwork is the result of such effort. It is a kernel process which emulates the performance bounded networks, and it acts as an integral module of the virtualization engine.

The vNetwork functions as an intermediate message broker between the vEntities pool and the *vMiddleware manager* as already seen in Fig. 4.6 and better highlighted with Fig. 4.12. Depending on the digital packet type and routing (source, destination) the appropriate network characteristics are applied to packets. As a matter of fact, there are three different type of messages illustrated in Fig. 4.12 with the three colored and numbered arrows.

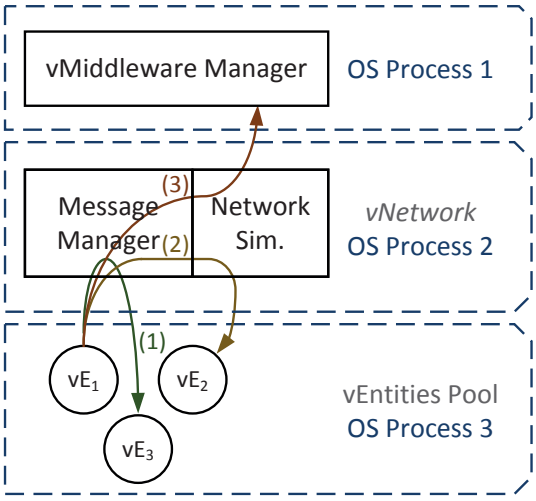


Figure 4.12 – The three domains of messages processed by the vNetwork with their paths. (1) internal, (2) network level, (3) building level

The `internal (1)` (cf. Fig. 4.12) messages are the ones exchanged between the vEntities. They are intended for internal communication purposes of the coupled models and thus are not emulated by the vNetwork. For example, a virtual PV panel model requires data from an external luminosity sensor to emulate the appropriate power. In reality, however, the power produced by the panel and the luminosity are physically associated due to the photovoltaic effect. As a result, no physical or virtual network is involved in this process.

The `network level (2)` (cf. Fig. 4.12) messages on the other hand are exchanged between control and sensing devices over their network. For example, a wireless motion sensor activates the wireless light bulbs when a user enters the room. This is the most frequently encountered communication type in the current fully integrated home automation systems.

Finally, the `building level (3)` (cf. Fig. 4.12) messages are exchanged between different networks, algorithms and systems in general. This type of interaction enables the next generation of adaptive SB. The vNetwork, following the emulation task, passes those types of packages to *vMiddleware manager* for uplink forwarding.

Developing a network simulator requires various simulation models and tools and is a complex process, both from the mathematical representation and the computational execution points of view. The authors in [218] attempt to overcome this limitation using a hybrid emulation architecture based on both physical and simulated network nodes. Additionally, numerous, mature network simulation tools exist such as ns-2 [219] and ns-3 [220], OMNeT++ [206], OPNET [221, 222], GloMoSim [223], BRITE [224] and SSFNet [225, 226]. Finally, literature also includes comparative studies between these tools in [227, 228, 229].

Fundamentally, network simulator extracts the statistical distributions that govern the simulated network topology and activity. There are various network parameters that can be

configured in those tools including, but not limited to, the following:

- Link capacity
- Bandwidth
- Response time
- Latency
- Routing protocol used
- Protocol overhead
- Frame size
- Dropped frame rate

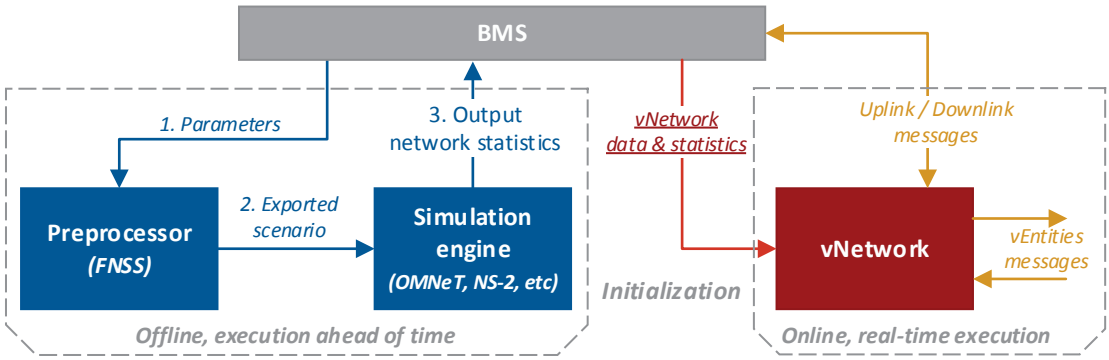


Figure 4.13 – The core workflow of the network emulation module of the building virtualization engine

The present work leverages the statistical analysis outputs of the tools mentioned above as an input for the vNetwork. The primary task of the vNetwork is to *enforce* the simulation tool's analyses on its own emulated networks. As it is illustrated in Fig. 4.13, two independent activity domains make up the vNetwork. The *offline activity* executes ahead of time, whereas the *online activity* runs continuously along with the virtualization engine.

The *offline activity* is responsible for extracting the network statistics based on configuration hosted on BMS servers and the use of dedicated network simulation tools. Fig. 4.13 highlights that process through a series of steps. Nonetheless, configuring a network simulation tool for a given topology and traffic scenario varies significantly depending on the simulation tool. Hence, a pre-processing step was introduced before invoking the network simulator. The FNSS [230] is a toolchain that can parse and generate various topologies, assign desired network parameters and generate traffic matrices or event schedules for various network simulators. The use of a pre-processing toolchain, like the FNSS, allows the vNetwork design to become independent from the network simulator employed.

More specifically, the *offline activity* execute in the following phases:

```
{
   "title": "NetworkSimulatorParameters",
   "type": "object",
   "properties": {
      "param_version": {
         "type": "number",
         "minimum": 1.0
      },
      "buffer_size": {
         "type": "integer",
         "minimum": 0
      },
      "link_capacity_bytes": {
         "type": "integer",
         "minimum": 1
      },
      "link_delay_ms": {
         "type": "integer",
         "minimum": 0
      },
      "topology_type": {
         "type": "string",
         "oneOf": [ "two_tier", "three_tier", "chord",
         "full_mesh", "line", "ring", "star" ]
      },
      "network_node_number": {
         "type": "integer",
         "minimum": 1
      },
      "traffic_schedule": {
         "type": "object",
         "oneOf":[
         {"fix_interval_ms" : {"type": "integer"} },
         {"poisson_avg_ms" : {"type": "integer"} } ]
      }
   }
}
```

Listing 7 – vNetwork parameters for network simulation

1. HTTP request to the BMS server for acquiring the initial network setup parameters such as: network node number, topology type, node send/receive buffer size, link capacity and traffic schedule. The response is a JSON formatted string validated by the schema in Listing 7.
2. The python based script uses the FNSS in order to extract the scenario for use in the dedicated network simulator.
3. Perform the offline simulation using the exported scenario and the network simulation tool. Retrieve the statistics required for the vNetwork which are the following:
   - packet loss
   - packet delay
4. Store the results of the analysis in the BMS database, thus characterizing each vEntity communication behavior.

Subsequently, the *online activity*, also visible in Fig. 4.13, operates in real time and enforces the

network particularities based on the data from the *offline activity*. Thus, it enables a real-time network emulation with acceptable accuracy even on the thinnest of hardware or numerous vEntities in the pool.

The packet loss probability for an emulated node is modeled with a Bernoulli distribution, a discrete probability distribution as a special case of the binomial distribution with the probability mass function (PMF) of Eq. 4.9. The packet is modeled as a successfully delivered with $n = 1$ or lost with $n = 0$ out of a single trial $N = 1$.

$$P_p(n|N) = \binom{N}{n} p^n (1-p)^{N-n}, n \in \{0,1\}, \quad N = 1 \tag{4.9}$$

The packet delays for an emulated node, depending on the network simulator output, can be:

- A mean delay in milliseconds to be applied to all the packets of the specific node.
- A common distribution such as normal, Poisson etc. that all delay samples follow.
- An output histogram of delays for each node. The random delays are drawn from each bin based on its calculated weight compared to the rest.
- A delay distribution described by a continuous PDF. In this case, the cumulative distribution function (CDF) should be initially obtained from the PDF. The inverted CDF when provided with uniform random numbers outputs the delays following that original PDF.

Following the presentation of vNetwork's system architecture, its software design and implementation are illustrated in Fig. 4.14. It consists of three threads in a single process. The vNetwork executes in a pipelined manner thanks to this multi-threaded design. The latter enables a higher throughput of packets compared to a non-pipelined execution. The choice of dedicated process for the vNetwork instead of sharing the vEngine one was taken for allowing a scale-up possibility. In that case, the latency sensitive vNetwork process can easily be spawned in different hardware without design reconsideration.

The network emulation cycle for each packet is broken down into a series of steps: receive, emulate loss, emulate delay, and forward, as depicted in Fig. 4.14. As long as each thread is processing every packet for about the same average time compared to others, the packet throughput is increased compared to a sequential algorithm. The packets pass through the vNetwork stages in a stream-like fashion. Each stage has an input, an output and a dedicated task for treating the packets according to the conditions derived from the source or destination of the packet.

Additionally, each stage has its time and data flow decoupled from the previous ones using asynchronous communication sockets (*Thread 1)*, thread-safe queues (*Thread 2*) or an

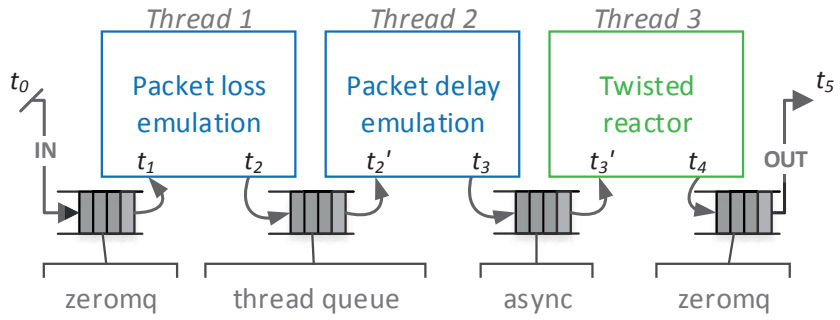


Figure 4.14 – Pipelined, triple-thread approach for zero additional time delay and high throughput

asynchronous callback framework (*Thread 3*). Unlike the synchronous pipelines where all the stages have to execute in-sync with the slowest among them (critical path), the interleaving of pipeline stages (threads) and asynchronous elements (sockets, queues, callbacks) enables an asynchronous pipelined execution. In it clear that the network emulation module has been designed to enable each thread/stage to produce and consume data packets as fast as it can without constraining its performance to the previous or next thread/stage.

The asynchronous elements permit a significant advantage over conventional pipelines where all the stages have to execute in time equal to the slowest among them amount. In hardware designs, it is achieved with slower clock speeds, whereas in software the previous stage cannot process a new input before the next one receives its output. Therefore, in the synchronous pipelines, the slowest stage eventually dictates the speed of the whole pipeline.

Algorithm 8 vNetwork threads initialization algorithm

```
1:  procedure INITIALIZE(n)
2:      if n = None then
3:          for all n ∈ networks do
4:              net_values[n] ← generate_values
5:              net_idx[n] ← 0
6:          end for
7:      else
8:          net_values[n] ← generate_values
9:          net_idx[n] ← 0
10:     end if
11: end procedure
```

More specifically, the *Thread 1* and *Thread 2* hold an array of numbers for each of the network they emulate. The arrays values depended on the scope of the thread and each emulated network; for example, it can be an array of packet delays or an array of 0 and 1 denoting the packet loss. Those arrays are populated during the start-up of the threads in order to minimize the latency and computational cost during the real-time operation. Each array is unique to the

Algorithm 9 Packet loss thread algorithm

```
1: procedure EMULATE_LOSS
2:     while loss_queue.len > 0 do
3:         pop the top of loss_queue to packet
4:         source ← source of packet
5:         n ← network index of source
6:         msg_success ← net_values[n][net_idx[n]]
7:         if msg_success then
8:             push packet to delay_queue
9:         end if
10:        net_idx[n] ← net_idx[n] + 1
11:        if net_idx[n] ≥ net_values[n].len − 1 then
12:            INITIALIZE(n)
13:        end if
14:    end while
15: end procedure
```

emulated network and the emulation task of each thread. The start-up initialization procedure in shown in Algorithm 8. The *n* is the network index and the *net_values* is the two-dimensional array. The first dimension denotes the network emulated and the second stores the arrays of the simulated output values over time. The algorithm for both *Thread 1* and *Thread 2* is similar except for the *generate_values* function.

To begin with, the *Thread 1* is populating the array using the `random.binomial` function, a member of the Python's numPy mathematical library. The function's argument "p" is the probability of packet success as generated by the network simulator when the number of trials equals one, as a special case of binomial distribution. The size of the array produced, function argument "size", is not strictly defined and varies based on the design requirements, packet quantity, and accuracy. The main procedure of the *Thread 1* is shown in Algorithm 9. As seen, when a packet arrives, the source network is identified and the value pointed by the *net_idx* index is taken from the relevant array. Depending on its boolean value, the packet is either dropped or pushed to the *Thread 2* queue. Then the index is increased pointing to the next value in the network values array. Thus, since the array is following the theoretical distribution, the packets would also do after a while. Lastly, if the index reaches the end of the array, a new one is generated with the index pointing to position 0.

The principle of operation of *Thread 2* is shown in Algorithm 10. The algorithm as expected is comparable with the one of *Thread 1*. The primary difference is in the way the array of delays (*net_values*) is generated. As it was mentioned earlier, unlike the packet loss algorithm which accepts a probability of packet success; the delay algorithm supports various statistical inputs and thus requires different procedures for the generation of the delay arrays.

In the first case where only a mean delay is provided, the array is populated only with this value

Algorithm 10 Packet delay thread algorithm

```
1: procedure EMULATE_DELAY
2:     while delay_queue.len > 0 do
3:         pop the top of delay_queue to packet
4:         source ← source of packet
5:         n ← network index of source
6:         ms_delay ← net_values[n][net_idx[n]]
7:         SEND_TO_TWISTED(packet, ms_delay)
8:         net_idx[n] ← net_idx[n] + 1
9:         if net_idx[n] ≥ net_values[n].len − 1 then
10:            INITIALIZE(n)
11:        end if
12:    end while
13: end procedure
```

and all packets are thus delayed equally.Secondly, where a typical distribution is provided, such as Gaussian or Poisson, the `random.normal` or `random.poisson` numPy functions are used respectively to generate the new random delays.

If instead a histogram or a sample of delays to be replicated are provided, the new random delays are drawn from each bin based on its calculated weight the `random.choice` numPy function. An example code snippet for achieving that functionality is seen in Listing 8.

```
import numpy as np

def random_by_hist(samples, n_bins=20):
   hist,bins = np.histogram(samples,bins=n_bins)
   hist=hist.astype(np.float32)
   weights=hist/np.sum(hist)
   new_random_samples=np.random.choice(bins[1:],len(samples),p=weights)
   return new_random_samples
```

Listing 8 – Generating random values from a given histogram

Finally, if a continuous PDF is provided, an inverse transform sampling should be performed. This is called non-uniform random variate generation. Devroye in [231] introduced the theorem which is used as the basis for generating the random numbers.

"***Theorem:*** *Let F be a continuous distribution function on $\mathbb{R}$ with inverse F^{-1} defined by $F^{-1}(u) = inf\{x : F(x) = u, 0 < u < 1\}$. If U is a uniform $[0,1]$ random variable then $F^{-1}(U)$ has distribution function F. Also, If X has distribution function F, then $F(X)$ is uniformly distributed on $[0,1]$*"

More specifically, the author at [232] elaborates on the exact procedure for inverse transform sampling. The steps to be followed are:

1. Normalize the given PDF, $f(x)$, if it is not already normalized; the PDF is normalized if $\int_{-\infty}^{+\infty} f(x)\,dx = 1$.
2. Integrate the normalized PDF in order to compute the CDF, $F(x) = \int_{-\infty}^{x} f(t)\,dt$.
3. Invert the $F(x)$ which results in the inverse CDF, $F^{-1}(u)$.
4. Generate a uniform random variable $U \in [0,1]$ and substitute it into the $F^{-1}(u)$.

The final *Thread 3*, cf. Fig. 4.14, is the Twisted reactor, a high-performance event-loop and the core of the Twisted network library. Although it is highly versatile as a tool, it serves a single purpose in this work. It has to handle a high number of packets and defer their transmission with minimum inherent latency. The Twisted reactor runs in the main thread of the process, and it is called from the packet delay module thread using `callFromThread` for thread-safe operation. Following that it schedules the delayed transmission of the packets using the `callLater` instruction. Finally, the txZMQ Python library closely integrates the ZeroMQ sockets inside the Twisted event loop for minimum additional latency and increased flexibility

Concluding this section, the evaluation and performance validation of the vNetwork module is in Section 4.6. It features several hosting hardware, packet throughput, and payload size scenarios.

4.6 Emulation Engine Evaluation and Validation

The previous sections presented the theory and implementation of the innovating building emulation engine based on the DES concepts. This section assesses the emulation engine performance in various case studies. Subsection 4.6.2 focuses on the vEngine while subsection 4.6.3 scrutinizes the vNetwork module performance. The section 4.6.4 concludes with a case study of virtualized components integration within existing infrastructure and building construction.

4.6.1 Testing setup

The aim of the work is a flexible architecture for different emulation scenarios. The requirements change not only along the emulation models but also concerning the preferred hosting hardware.

A setup was designed and tested for evaluating the emulation engine design and operation on many realistic scenarios. The setup is comprised of:

1. hosting hardware for the computational execution;
2. modified Debian Linux distribution for software and library support;
3. custom software logic that supervises the evaluation process and automates the measurements;

4. custom algorithms for statistical analysis and plotting of the measurements;
5. the enabling BMS for enabling the CPS properties;
6. and finally, the source code of the emulator.

The diversified hardware enables the evaluation of dissimilar use cases such as: as an offline simulation tool, as an online and cloud-hosted emulator or even by mean of building-distributed micro-emulators. The representative hardware fitting these scenarios are the following:

1. powerful modern machine, featuring an Intel® Core i7-6700 CPU @ 4.00Ghz with 32GB DDR4 memory;
2. mainstream hardware, featuring an Intel® Core i5-4570 CPU @3.60Ghz with 8GB DDR3 memory;
3. last generation hardware, featuring an Intel® Core 2 Quad Q9650 @3.00Ghz with 8GB DDR2 memory;
4. an inexpensive, cloud-hosted virtual private server (VPS), featuring a shared Intel® E5-2630L CPU and 512MB RAM;
5. a capable and last generation micro-computer, the Raspberry Pi 3 (Rasp3), featuring a quad-core ARM® Cortex-A53 MPU @1.2Ghz with 512MB LPDDR2 memory;
6. finally, an industrial, embedded, low-power Linux board, the BeagleBone Black (BBB), featuring a single-core ARM® Cortex-A8 MPU @1Ghz with 512MB of LPDDR3 memory.

It is worth noting that due to memory size limitations, the machines equipped only with 512MB of RAM cannot support more than 100 vEntities. Hence, the VPS, Raspberry, and BeagleBone related graphs in the following subsections do not include the tests of more than 100 vEntities.

To assess and normalize the hardware regarding performance capacity, the open-source `sysbench` utility have been used. While it supports several tests, for this work, only the CPU and RAM had been tested. Those two components, unlike the storage IOPS, significantly influence the performance of the emulation engine. Specifically, a series of three tests, for each hardware, has been performed.

1. Single-threaded CPU benchmark; the benchmark consists in timing the calculation of prime numbers up to 10000.
2. Multi-threaded CPU benchmark; the same configuration with the previous but using 8 threads.
3. RAM speed benchmark; single-threaded, sequential write test for 1 GByte of data with 1 KByte block size.

Listing 9 displays the exact `sysbench` arguments that had been used. The results of the test are found in the Table 4.1. The results provide helpful preliminary insights on the anticipated

performance of the engine. Firstly as expected, the multi-core architectures significantly benefit from multi-threaded computations. As the emulation engine is also a multi-threaded architecture, the benefits are assumed to appear also in the emulation related benchmarks. Secondly, while the memory speeds varied significantly, the realistic engine results did not reveal any strong correlation between memory speed and emulation performance. Finally, the computational heavy, multi-threaded benchmark exhibits a significant contrast in performance (e.g. $\approx 252:1$ for i7-6700 vs BeagleBone) of the hardware; this enables the evaluation of the engine on the two extreme ends of hardware spectrum.

```
#!/bin/bash
#Benchmark the CPU performance
sysbench --test=cpu run

#Benchmark the threaded CPU performance
sysbench --test=cpu --num-threads=8 run

#Benchmark the memory performance
sysbench --test=memory --memory-total-size=1G run
```

Listing 9 – Bash script for hardware benchmark

Table 4.1 – Benchmark metrics using sysbench for the hardware used in the performance evaluation of emulation. *For CPU smaller is better, for Memory bigger is better.*

Hardware Configuration	Sysbench Results		
	CPU 1-thread	CPU 8-threads	Memory
Intel® i7-6700, 32GB DDR4	7.35 *sec*	1.15 *sec*	3990 *MB/sec*
Intel® i5-4570, 8GB DDR3	8.6 *sec*	2.28 *sec*	3830 *MB/sec*
Intel® Q9650, 8GB DDR2	8.4 *sec*	2.12 *sec*	2250 *MB/sec*
Intel® E5-2630L *(Shared)*, 512MB DDR3	12.35 *sec*	12.84 *sec*	692 *MB/sec*
Raspberry Pi 3, 512MB LPDDR2	182.6 *sec*	45.72 *sec*	318 *MB/sec*
BeagleBone Black, 512MB LPDDR3	289.4 *sec*	289.8 *sec*	155 *MB/sec*

To normalize the testing hardware for the software point of view, a clean installation of the Debian "jessie" has been used. Additionally, all unnecessary background services had been suspended during the tests. This step is necessary for reducing externally induced variance on the results.

The majority of the results in the following subsection are in the form of CDF diagrams. They have several advantages over histograms. Firstly, all the key values like minimum and maximum, median and percentiles can be directly read from the diagram. Histograms illustrate the minimum and maximum of the samples as values in the first or last bin accordingly. On the contrary, the minimum is the CDF diagram is the point where the curve meets the x-axis, while the maximum is where it reaches the $y = 1$. The percentiles can easily be read using the x-axis. Secondly, outliers in histograms stretch the bins and make it difficult to recognize distribution

patterns quickly. The outliers for the CDF on the other hand can be seen through the tails of the curves. While harder than with histograms, the clusters of values can be read from the CDF diagrams as well. A decrease of the curve slope followed with an increase again denotes a group of samples with values read on the x-axis. Finally and most importantly for the scope of this section, the CDF diagrams are much more suitable for comparison of several datasets. An arbitrary number of CDF curves can be plotted in the same figure for direct comparison.

4.6.2 vEngine performance

To begin with, in high activity periods of the vEntities pool due to the cooperative design of the emulation engine, computation and communication overhead may be encountered. During these demanding times, sockets have to receive and transmit hundreds to thousands of messages per second. Despite the regulation services by the *supervisor* module, with this subsection, the author validates the design against unfavorable conditions. The section illustrates the capability of the engine to cope with a large number of concurrent computation and communication activities.

A special purpose vEntity called vBenchmark is implemented for evaluating that capability. Unlike the various vEntities presented in subsection 4.5.3, this model class allows precise configuration of the activity patterns. Examples of configurable elements include the message size (L), the sleep unless interrupted interval (Δt_i), the time interval of uninterpretable computations (Q_i), and finally the frequency of generated events (F^{evt}) and received commands (F^{cmd}). Additionally, the number (n) of vBenchmark-type of vEntities running concurrently is also adjustable.

More specifically, to isolate and study the inherent overhead of the engine rather than of the models, the computational time Q_i parameters for all the vBenchmarks was set to zero. In that case, the main workload W_p is caused by the various sockets (cf. Fig. 4.6) in the engine. The workload over a period of time $[t_0, t_1]$ is calculated by Eq. 4.10, where $F^x(t)$ is the frequency of messages over time, $L^x(t)$ the size of message over time. The superscript x denotes the type of the message.

$$W_p = \int_{t_0}^{t_1} L^x(t) \cdot F^x(t)\, dt, \quad x \in \{evt, cmd\} \;\; \forall\, zmq\ socket \tag{4.10}$$

The workload of each vEntity is given by Eq. 4.11. The $L_i^{evt}(t)$ and $F_i^{evt}(t)$ functions define the events from vE_i. The $L_i^{cmd}(t)$ and $F_i^{cmd}(t)$ functions characterize the external commands

from BMS to the specific vE_i.

$$\begin{aligned} W_{vE_i} &= W_{vE_i}^{evt} + W_{vE_i}^{cmd} \\ &= \int_{t_0}^{t_1} L_i^{evt}(t) \cdot F_i^{evt}(t)\, dt + \int_{t_0}^{t_1} L_i^{cmd}(t) \cdot F_i^{cmd}(t)\, dt \end{aligned} \tag{4.11}$$

The workload of the complete pool on the other hand is defined by Eq. 4.12 as a sum of all vEntities.

$$W_{vE}^{all} = \sum_{i=1}^{n} W_{vE_i} \tag{4.12}$$

Combining Eq. 4.12 and Eq. 4.11 results in the total workload of the vEntities pool given by Eq. 4.13.

$$\begin{aligned} W_{vE}^{all} &= \sum_{i=1}^{n} W_{vE_i}^{evt} + \sum_{i=1}^{n} W_{vE_i}^{cmd} \\ &= \sum_{i=1}^{n} \int_{t_0}^{t_1} L_i^{evt}(t) \cdot F_i^{evt}(t)\, dt + \sum_{i=1}^{n} \int_{t_0}^{t_1} L_i^{cmd}(t) \cdot F_i^{cmd}(t)\, dt \end{aligned} \tag{4.13}$$

The workload of *vMiddleware manager* sockets is defined as the sum of workloads at events and commands sockets as illustrated by Eq. 4.14.

$$W_{vMid} = W_{vMid}^{evt} + W_{vMid}^{cmd} \tag{4.14}$$

Specifically, Eq. 4.15 defines the workload on events sockets, where p denotes additionally the probability of a message to be forwarded to BMS instead of being forwarded to other entities (internal message).

$$W_{vMid}^{evt} = W_{re}^{evt} + p \cdot W_{fw_{BMS}}^{evt} + (1-p) \cdot W_{fw_{pool}}^{evt}, \qquad p \in [0,1] \tag{4.15}$$

Since the workload at any socket is the same, using Eq. 4.10, the events workload on the receive and forward sockets of the *vMiddleware manager* are given by Eq. 4.16.

$$\begin{aligned} W_{re}^{evt} &= W_{fw_{BMS}}^{evt} = W_{fw_{pool}}^{evt} = \sum_{i=1}^{n} W_i^{evt} \\ &= \sum_{i=1}^{n} \int_{t_0}^{t_1} L_i^{evt}(t) \cdot F_i^{evt}(t)\, dt \end{aligned} \tag{4.16}$$

Additionally, Eq. 4.17 defines the workload on commands sockets, where $W_{re_{BMS}}^{cmd}$ relates to commands reception from BMS and $W_{fw_{pool}}^{cmd}$ to their forwarding to entities pool.

$$W_{vMid}^{cmd} = W_{re_{BMS}}^{cmd} + W_{fw_{pool}}^{cmd} \tag{4.17}$$

For $L_{BMS}^{cmd}(t)$ and $F_{BMS}^{cmd}(t)$ functions which characterize the commands from BMS to any vE_i:

$$W_{re_{BMS}}^{cmd} = W_{fw_{pool}}^{cmd} = \int_{t_0}^{t_1} L_{BMS}^{cmd}(t) \cdot F_{BMS}^{cmd}(t)\, dt \tag{4.18}$$

Therefore, combining Equations 4.15, 4.16, 4.17, 4.18 results in the final *vMiddleware manager* given by Eq. 4.19

$$\begin{aligned} W_{vMid} &= W_{re_{pool}}^{evt} + p \cdot W_{fw_{BMS}}^{evt} + (1-p) \cdot W_{fw_{pool}}^{evt} + W_{re_{BMS}}^{cmd} + W_{fw_{pool}}^{cmd} \\ &= 2 \cdot W_{re\_fw}^{evt} + 2 \cdot W_{re\_fw}^{cmd} \\ &= 2 \cdot \sum_{i=1}^{n} \int_{t_0}^{t_1} L_i^{evt}(t) \cdot F_i^{evt}(t)\, dt + 2 \cdot \int_{t_0}^{t_1} L_{BMS}^{cmd}(t) \cdot F_{BMS}^{cmd}(t)\, dt \end{aligned} \tag{4.19}$$

Concluding the workload calculations, the total workload of the virtual engine, W_{vEng}, over a

period of time $[t_0, t_1]$ is given by Eq. 4.20.

$$\begin{aligned} W_{vEng} &= W_{vE}^{all} + W_{vMid} \\ &= 3 \cdot \sum_{i=1}^{n} \int_{t_0}^{t_1} L_i^{evt}(t) \cdot F_i^{evt}(t)\, dt + \sum_{i=1}^{n} \int_{t_0}^{t_1} L_i^{cmd}(t) \cdot F_i^{cmd}(t)\, dt \\ &+ 2 \cdot \int_{t_0}^{t_1} L_{BMS}^{cmd}(t) \cdot F_{BMS}^{cmd}(t)\, dt \end{aligned} \tag{4.20}$$

To quantify the performance, the author chose the *roundtrip* latency as the key metric. This latency measures the total delay for a message to be acknowledged by the vEntity (commands-relevant) and for the *vMiddleware manager* to receive the reply (events-relevant). The delay for a vEntity to acknowledge the message is due to the cooperative nature of the vEntities pool. It had been already mentioned in the subsection 4.5.4 of the supervisor as $d_i = \Delta t_i^a - \Delta t_i$. On the other hand, the delay of the *vMiddleware manager* to acknowledge the reply, expresses the overall load of the engine manager that is responsible for numerous other tasks besides the vEntities events reception.

Fig. 4.15 to 4.19 illustrate the evaluation results using the customized vBenchmark class. Each curve or bar in the figures represents a single test. The colors in each figure represent a different size of vEntities pool as defined by their legend.The test executes until a hundred of commands and equal number events per vEntity have been collected. While the *roundtrip* latency is the dependent variable of the statistical analysis, the independent variables are:

1. commands/sec, denotes the frequency of generated messages (commands) by the *vMiddleware manager* addressed to a single vEntity;
2. events/sec, denotes the frequency of generated messages (events) by each vEntity;
3. number of vEntities, as the name suggests, it is the number of parallel uTreads;
4. hardware, indicates the hosting hardware as described above.

Starting with Fig. 4.15, it represents a low activity scenario for the emulation engine. The middleware receives a command every 10 seconds, while each vEntity sends an event also every 10 seconds. In this scenario, it is obvious that the number of uTreads do not influence the performance of the engine. For the capable hardware of Fig. 4.15(a)-(d) the latencies are proven to be superb and within the timing error, even with $\leq 1ms$ latency. For the embedded hardware of Fig. 4.15(e)-(f) the latencies are also excellent for any potential use-case.

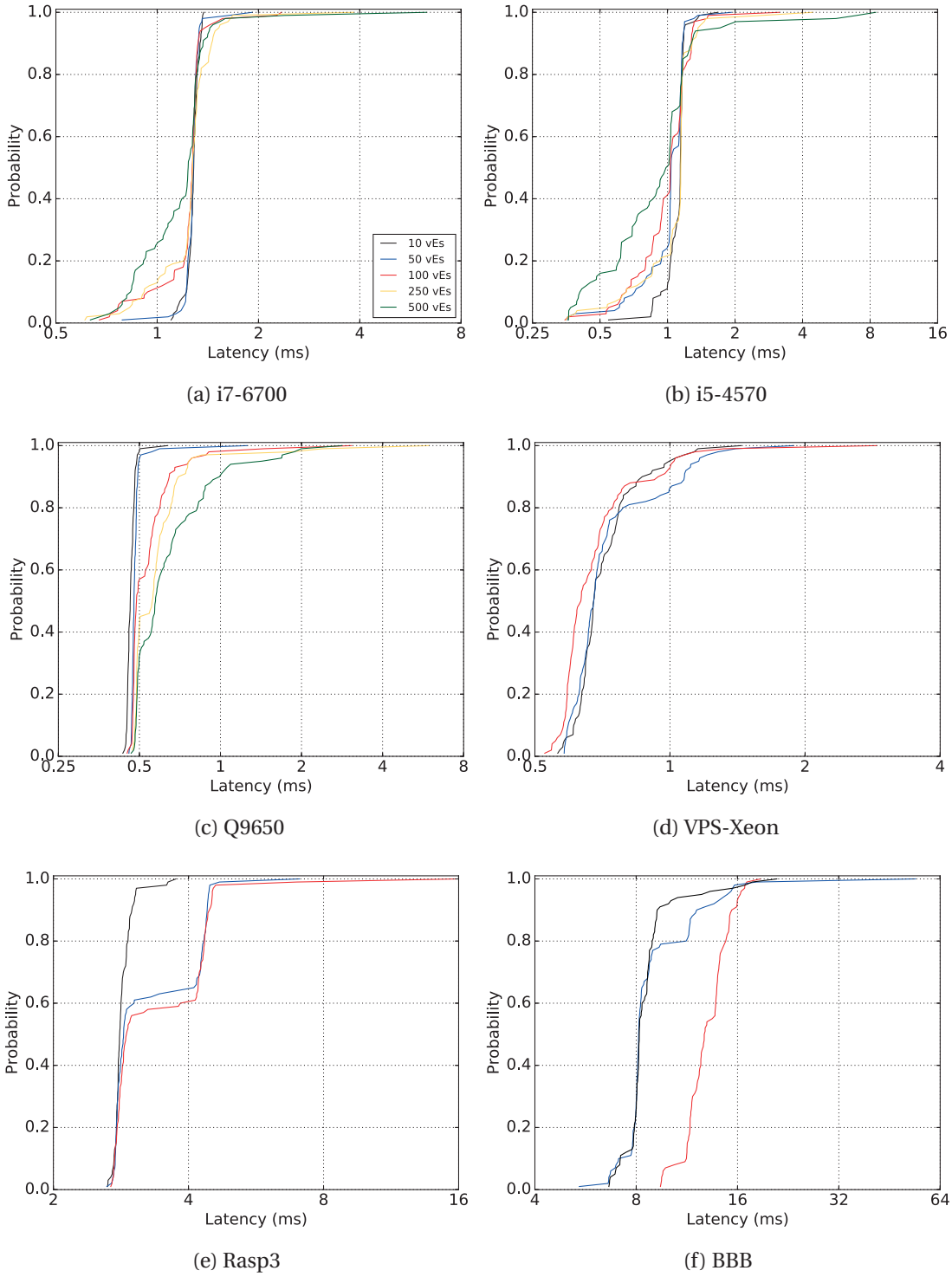
Fig. 4.16 displays the cluster of tests that increases the commands activity to 10 commands per second while keeping the events at the same frequency. Despite the increase in the activity on the *vMiddleware manager* side due to greater command workload, the *roundtrip* latency for all the tests remain unaffected.

Fig. 4.17 displays the cluster of tests with increased frequency of events at 100 per second while the commands remain at the modest number of one every 10seconds. This level of activity is understandably fairly unrealistic in real-life deployment. Nobody could envision a lot of building infrastructure that produces hundreds of events per seconds. Nevertheless, this scenario is critical for evaluating the absolute limits of the engine. Unlike the unobserved impact of increased commands activity in the previous figure, the number of events certainly affects the real-time operation of the engine. To begin with, while the performance in Fig. 4.17(a)-(e) remains excellent, the advantage of the superior hosting hardware becomes noticeable. In fact, BeagleBone in 4.17(f) shows some considerable delays when loaded with more than 50 vEntities.

Fig. 4.18 are the last CDF figures illustrating the vBenchmark tests. The figures feature the maximum activity scenario tested, 100 events per seconds and 10 commands per second. The observation of previous test are confirmed with this figures as well. Similar to Fig. 4.16, the increase of commands frequency does not affect the performance of the engine. On the other hand, similar to Fig. 4.17, the increase of events per seconds considerably impacts the real-time operation.

Fig. 4.19 illustrates a different data analysis. The latency is separated now in event-related (evt) and command-related (cmd) ones. That analysis highlights and facilitates the identification of the message type that contributes in the increased latency under high activity scenarios. The amplitude of the histograms represents the mean value of the samples population, while color depicts the number of vEntities. Each subfigure relates to the previous scenarios of Fig. 4.15 to 4.18. It is apparent that the cause of the *roundtrip* latency is on command direction of the message; meaning that the message is queued at the vEntities socket until it has the chance to acknowledge and timestamp it. Obviously, a raise in a number of active vEntities sharing the same process increases the chance of overlapping requests for CPU that need to be queued and sequentially served.

It is important to notice that the memory size, rather than its speed, is crucial for a large number of emulated infrastructure. For example, the VPS has still computational capacity for more uTreads while being limited by its memory size. The considerable memory is due to the loaded vEntity class objects. Even though only one model (for each running engine) executes at a time, their data structures, variables, and program code are kept and not purged every time the control of uTread is relinquished. Hence, as vBenchmark is a relatively simple model, the effects of limited memory will be more prominent for more advanced models. Nevertheless, due to a single uTread memory stack being active at any point, long-suspended vEntities can be swapped and memory reused by other processes. However, this paging operation by the memory management unit will undeniably introduce delays that need to be further investigated. Finally, for hardware architecture with many small cores, e.g. Raspberry Pi 3, a second parallel vEntity pool can be spawned as an additional process. The uTreads are then shared between the two. This trivial procedure enables improved utilization of multiple CPU cores.



(a) i7-6700

(b) i5-4570

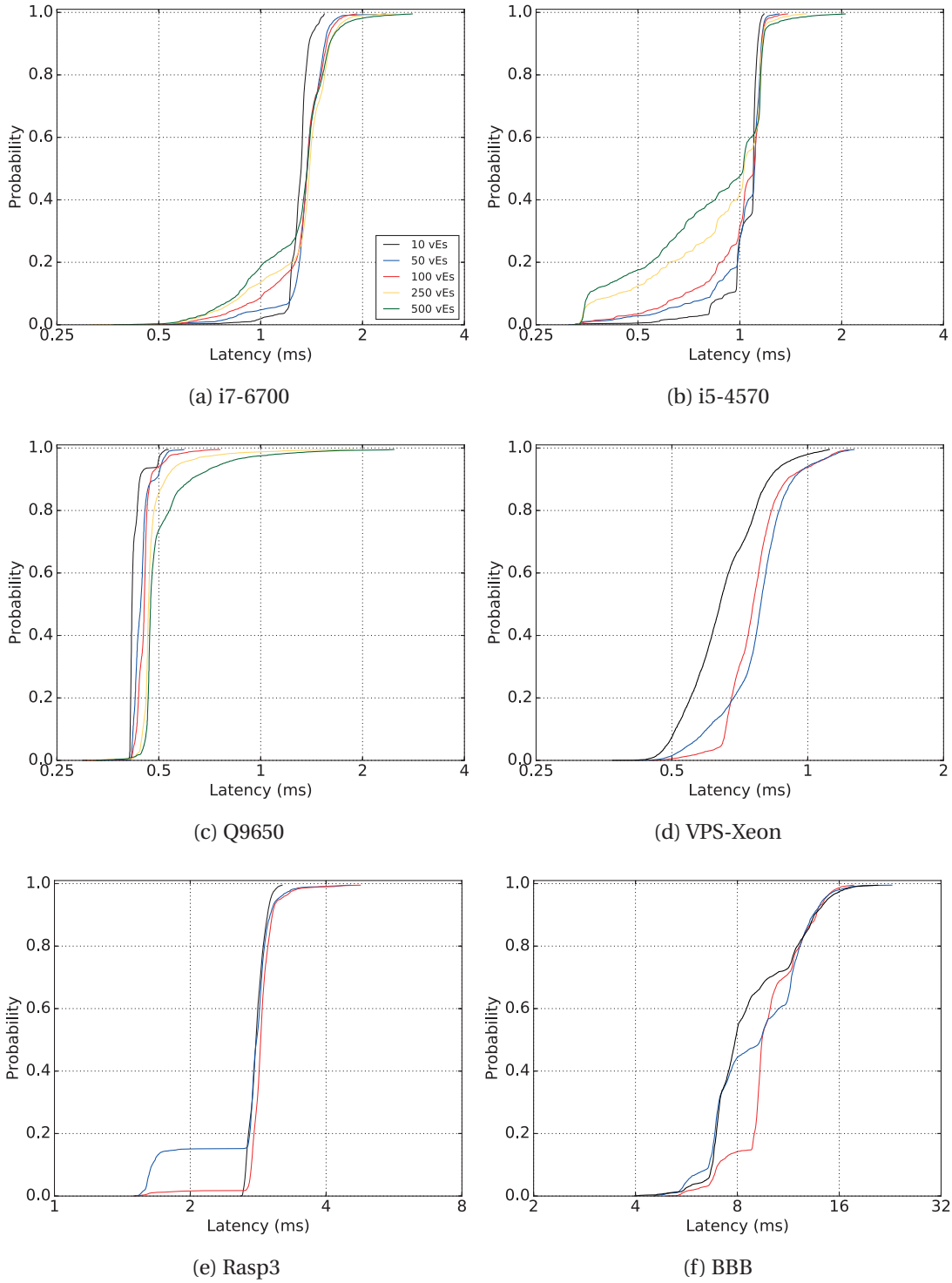
(c) Q9650

(d) VPS-Xeon

(e) Rasp3

(f) BBB

Figure 4.15 – CDF of roundtrip latency, for varying number of vEntities and hardware, for 0.1 commands/sec and 0.1 events/sec for each vEntity



(a) i7-6700

(b) i5-4570

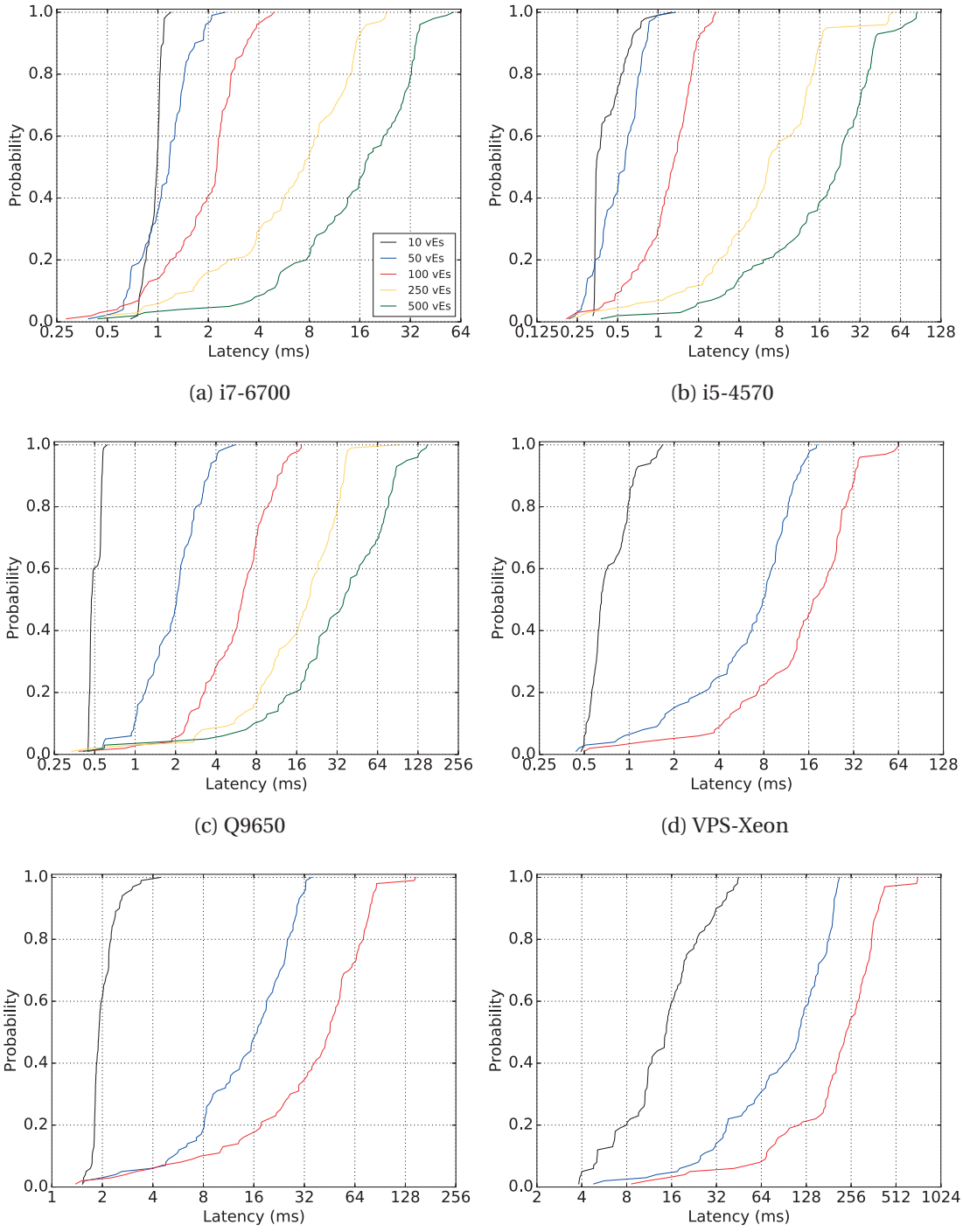
(c) Q9650

(d) VPS-Xeon

(e) Rasp3

(f) BBB

Figure 4.16 – CDF of roundtrip latency, for varying number of vEntities and hardware, for 10 commands/sec and 0.1 events/sec for each vEntity



(a) i7-6700

(b) i5-4570

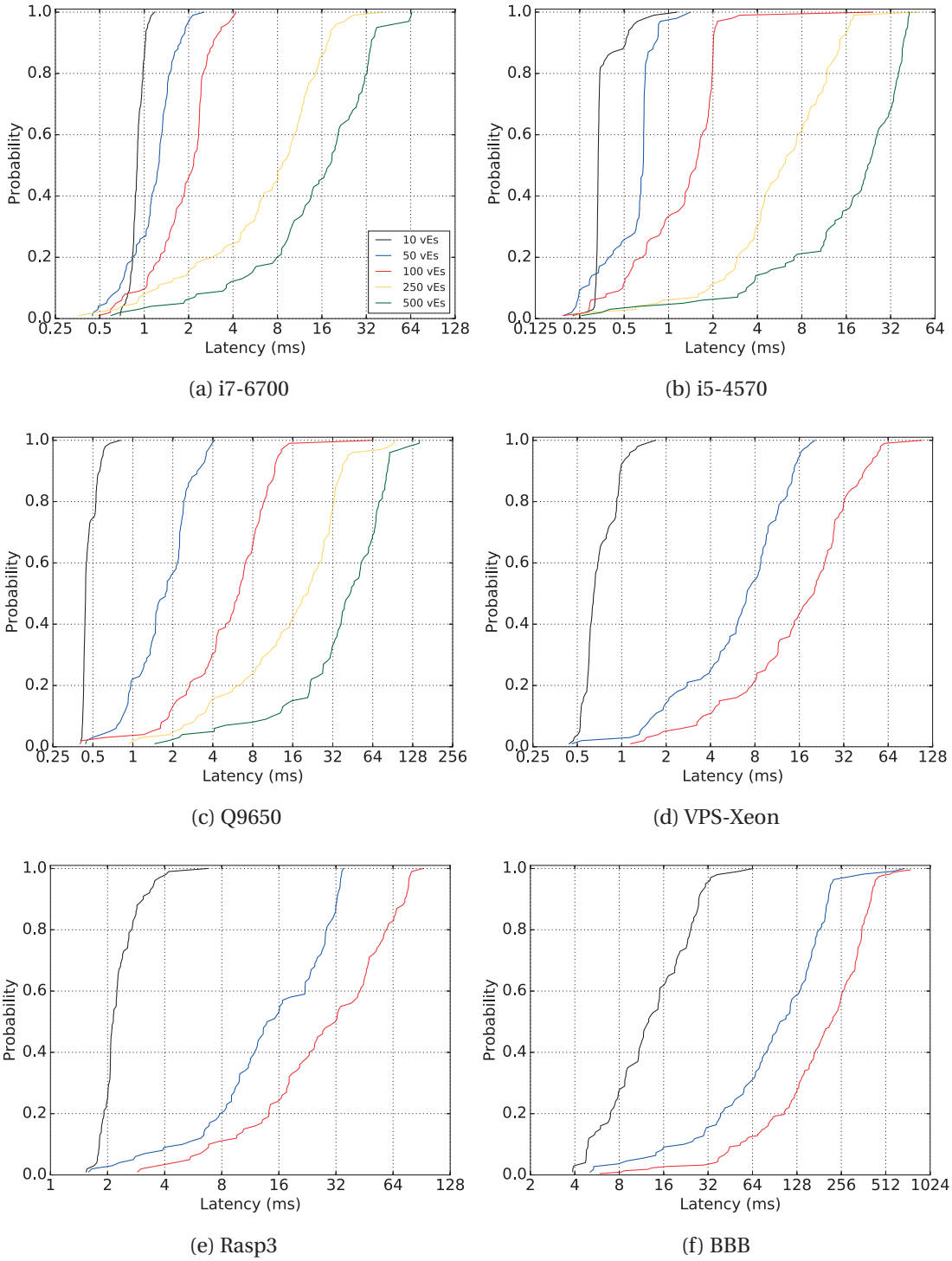
(c) Q9650

(d) VPS-Xeon

(e) Rasp3

(f) BBB

Figure 4.17 – CDF of roundtrip latency, for varying number of vEntities and hardware, for 0.1 commands/sec and 100 events/sec for each vEntity



(a) i7-6700

(b) i5-4570

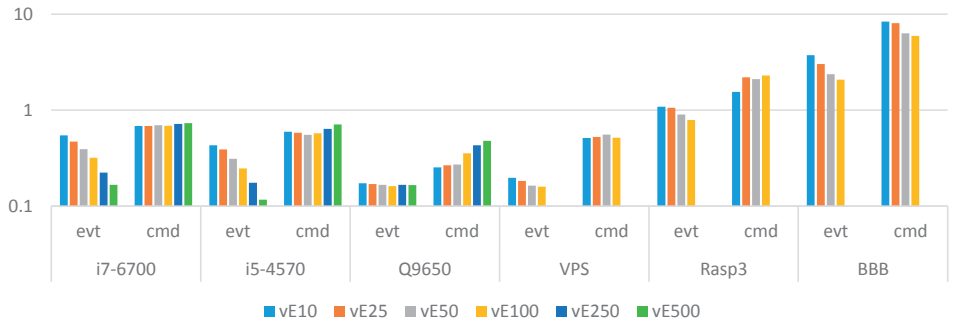
(c) Q9650

(d) VPS-Xeon

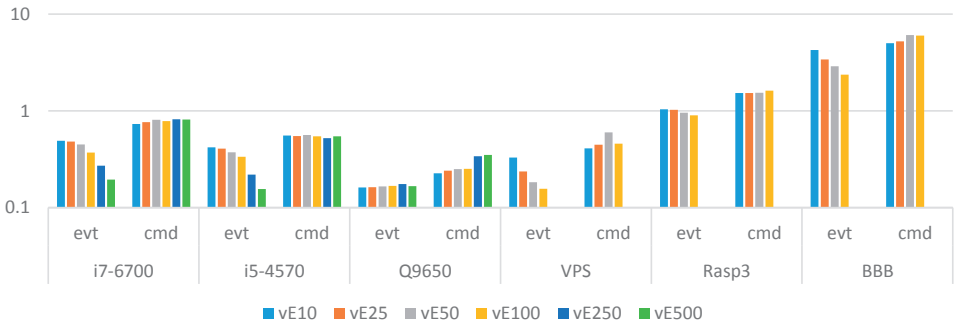
(e) Rasp3

(f) BBB

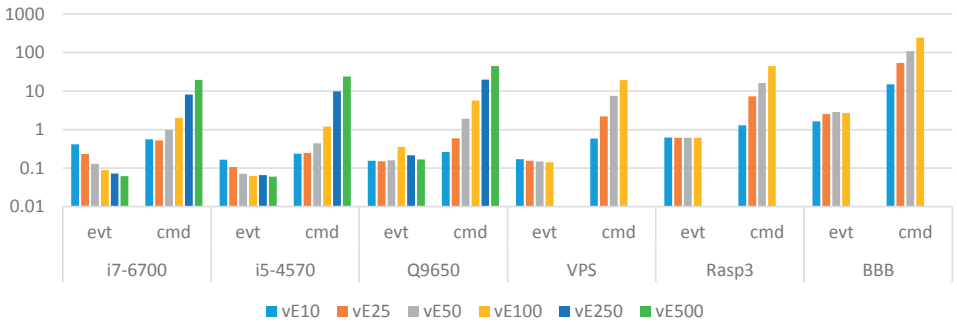
Figure 4.18 – CDF of roundtrip latency, for varying number of vEntities and hardware, for 10 commands/sec and 100 events/sec for each vEntity



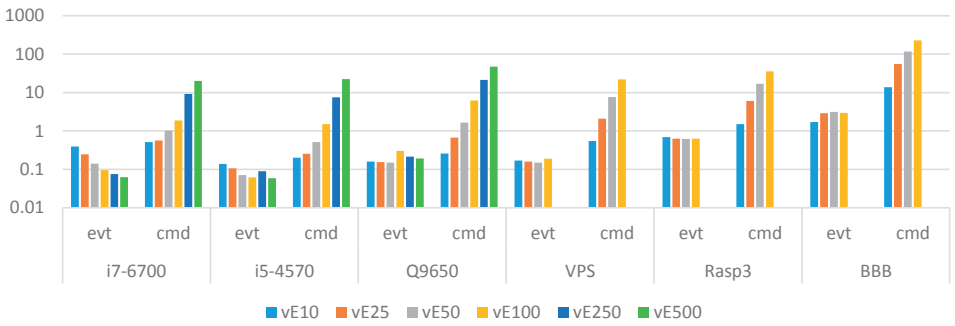
(a) 0.1 commands/sec 0.1, events/sec



(b) 10 commands/sec 0.1, events/sec



(c) 0.1 commands/sec 100, events/sec



(d) 10 commands/sec 100, events/sec

Figure 4.19 – Isolated latency of events and commands for varying hardware, number of vEntities, commands/sec, and events/sec

4.6.3 vNetwork performance

The subsection 4.5.5 introduced the design and implementation of the vNetwork module. Similar to the core engine, the cooperative nature of the design does not enforce the execution timings. The vNetwork's pipeline follows a "best effort" approach. Hence, in situations of a large number of emulated nodes with high messaging activity, moments of congestion and increased latency could arise. Therefore, an evaluation of vNetwork's performance under load was carried out. Similarly to the engine performance analysis, some variables have been used for evaluating the vNetwork module. Unlike the vBenchmark scenarios though, in vNetwork there is not differentiation between events or commands since its pipeline is unidirectional, cf. Fig. 4.14.

Since vNetwork module emulates real networks, a metric of interest is the throughput B_t of processed packets as defined by Eq. 4.21 where N_t denotes the number of packets. The steady state throughput B is calculated by Eq. 4.22 [233].

$$B_t = \frac{N_t^{total}}{t} = \frac{N_t^{evt} + N_t^{cmd}}{t} \tag{4.21}$$

$$B = \lim_{t\to\infty} B_t = \lim_{t\to\infty} \frac{N_t^{evt} + N_t^{cmd}}{t} \tag{4.22}$$

The mean size of the packet is given by Eq. 4.23 where L_i the size of each packet excluding the zmq header.

$$L_p = \frac{\sum_{i=1}^{N_t} L_i}{N_t} \tag{4.23}$$

The W_{vNet}, given by Eq. 4.24, is the average workload due to the two, receive and forward, sockets of the vNetwork pipeline. The internal computational work of *vNetwork* is negligible. The p_l denotes the probability of a packet to be emulated as "lost" and thus not to require submission, W_{re} relates to the reception socket and W_{fw} to the forward one.

$$W_{vNet} = W_{re} + (1 - p_l) \cdot W_{fw} \tag{4.24}$$

Given that the workload per socket is defined by

$$W_{re} = W_{fw} = L_p \cdot B \tag{4.25}$$

the total workload of vNetwork becomes:

$$\begin{aligned} W_{vNet} &= L_p \cdot B + (1 - p_l) \cdot (L_p \cdot B) \\ &= 2 \cdot L_p \cdot B - p_l \cdot L_p \cdot B, \qquad p \in [0,1] \end{aligned} \tag{4.26}$$

Similarly to the vBenchmark subsection, in order to assess the performance the author tested the vNetwork versus a number of variables such as:

- incoming packet frequency;
- packet size;
- hosting hardware.

Fig. 4.20 to 4.25 and Table 4.2 illustrate the performance of vNetwork on different evaluation scenarios. Specifically, there were performed two series of tests for the vNetwork module. The first one involves a fixed packet size and varying packet throughput, while the second uses varying packet size and fixed packet throughput. Additionally, both of them have been repeated on the six hosting hardware of the previous subsection. To yield accurate results, a population of 50000 latency samples, for each test, has been collected for the statistical analysis.

Unlike the vEntities benchmark, where a degree of latency can be tolerated, for vNetwork it is not the case. The emulated networks have inherent latencies of only a few dozens of ms. Hence, the delay introduced by the vNetwork module should be at least one order of magnitude smaller. It is therefore obvious that the real-time performance of the network emulator is critical for the overall accuracy of the engine.

Starting with Fig. 4.20, the network emulation module is evaluated with a fixed message payload size of 100 B with packet frequency ranging from 1 packets/sec to 2500 packets/sec. Each colored curve represents a different packet frequency. As the figure shows, for this payload size, the powerful machines, Fig. 4.20(a)-(c) encounter no trouble whatsoever, with most packets traversing the pipeline with latency of $\leq 1ms$. The cloud-based solution, as well as the Raspberry Pi 3, are introducing minimal latency for nearly all the packet throughputs. An exaggerated situation in pictured in Fig. 4.20(f) for the BeagleBone which is able to sustain only up to 100 packets/sec.

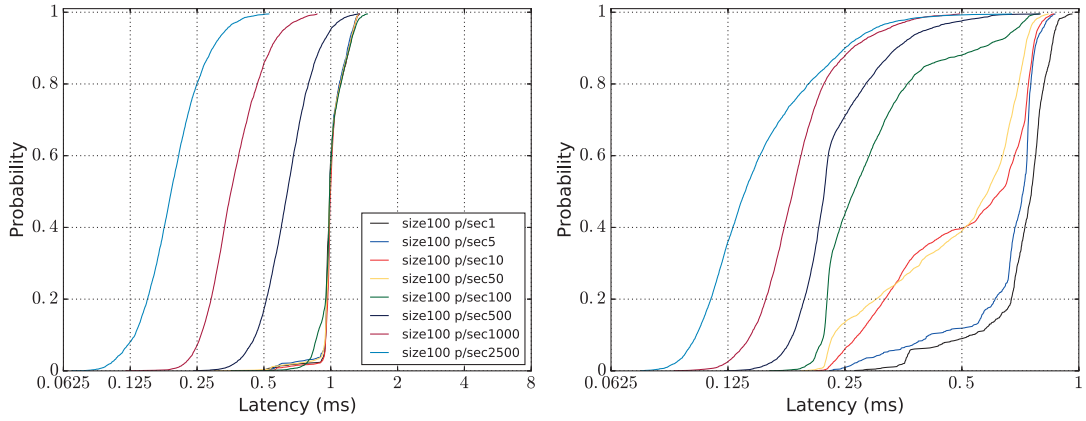
The previous test of varying packet throughput is repeated for a payload size of 500 B. It permits the performance assessment when emulating large frame network protocols. The resulting CDF diagrams are visible in Fig. 4.21. Even with the increased payload, large core architectures can cope with the increased throughput, cf. Fig. 4.21(a)-(d), with only exception the case of 2500 packets/sec on the cloud VPS. The quad-core Raspberry Pi maintained surprisingly good performance even for 2500 packets/sec. The BeagleBone's limitations are also appearing in this scenario, necessitating less than 100 packets/sec to operate in near real time.

Fig. 4.22 and 4.23 illustrate scenarios with fixed frequency and varying payload size. Those tests enable the study of packet size, rather than throughput, impact on the operation of the engine. More specifically, the two tests now feature a fixed packet frequency of either 100 or 500 packets/sec and a varying payload size of 5 B to 10 KB. It is worth noting that payloads ≥ 1 KB are rather unrealistic for the relevant device networks in the building, and they serve only the assessment process.

Fig. 4.22(a)-(d) reveals the great performance regardless of the payload size for all the large core architecture, even for the shared CPU core of the VPS. While not in the same performance scale, the Raspberry Pi 3 can sustain the performance with up to 5 KB payload. On the other hand, the BeagleBone can maintain the same performance with up to 1 KB payload.

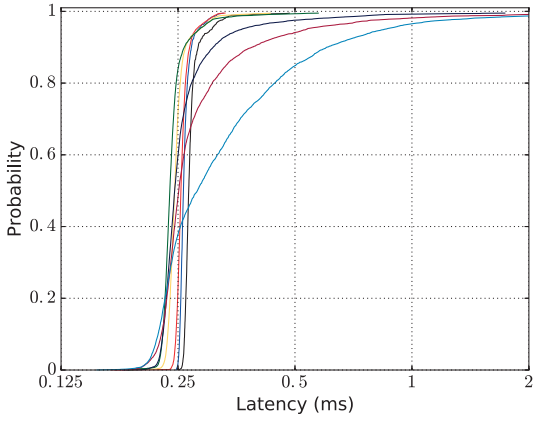
Fig. 4.23 reveals a different situation. While the three first large core architectures, cf. Fig. 4.23(a)-(c), are not affected by the increased throughput, that is not the case for the less powerful hardware, cf. Fig. 4.23(d)-(f). The VPS and Raspberry Pi 3 start to show signs of performance degradation from 5 KB payload. Even worse, the BeagleBone is unable to provide real time network emulation even for payloads as low as 5 B. This proves in fact that for vNetwork module, real time operation highly depends on the packet throughput capabilities of the hardware.

For holistic comparison reasons Table 4.2 is provided. It includes the mean and standard deviation statistical data for all the tests conducted during the evaluation of vNetwork. The table middle-rules separate the four test series as mentioned above, while the color denotes the unacceptable latencies for realistic network emulation.

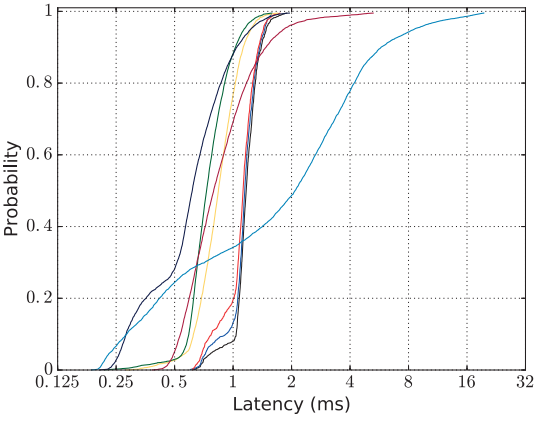


(a) i7-6700

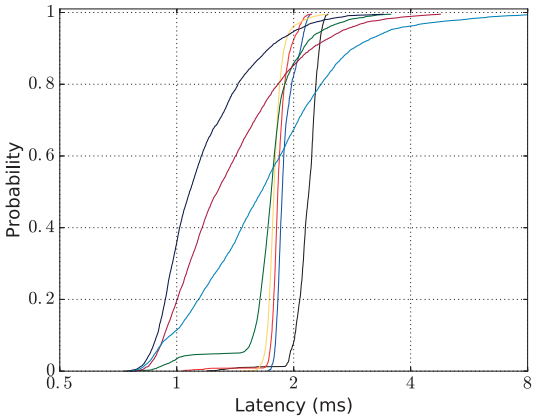
(b) i5-4570



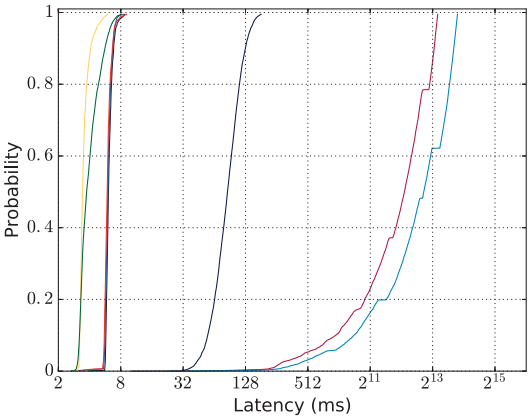
(c) Q9650



(d) VPS-Xeon

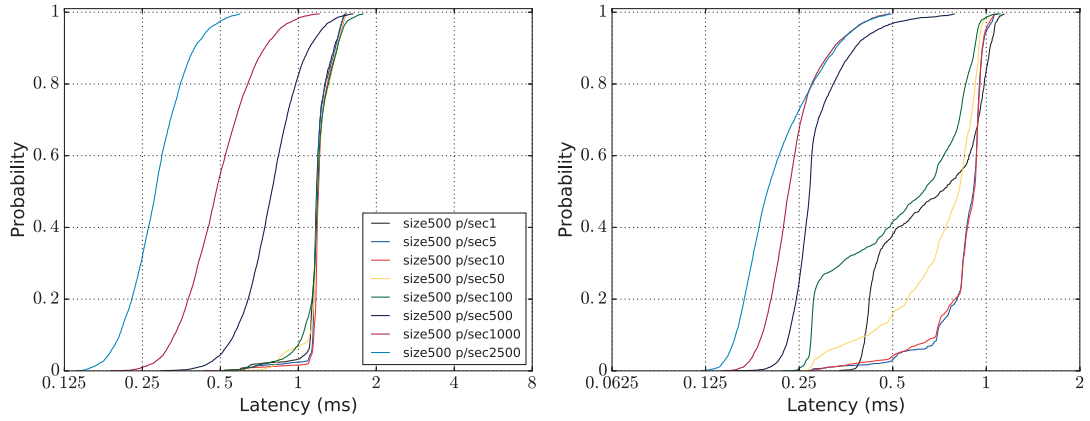


(e) Rasp3



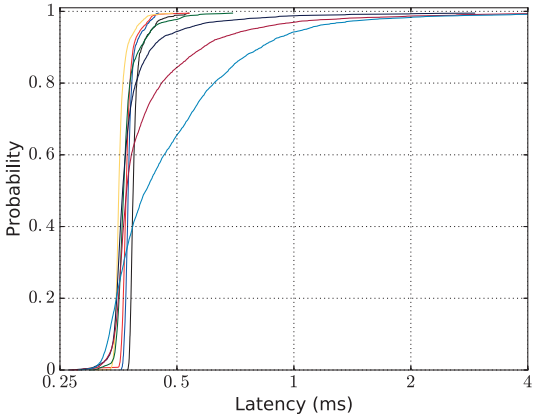
(f) BBB

Figure 4.20 – CDF of latency introduced by vNetwork hosted on varying hardware, for varying packet/sec and 100 B payload

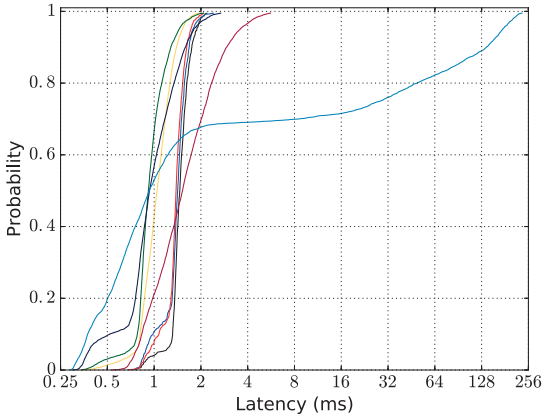


(a) i7-6700

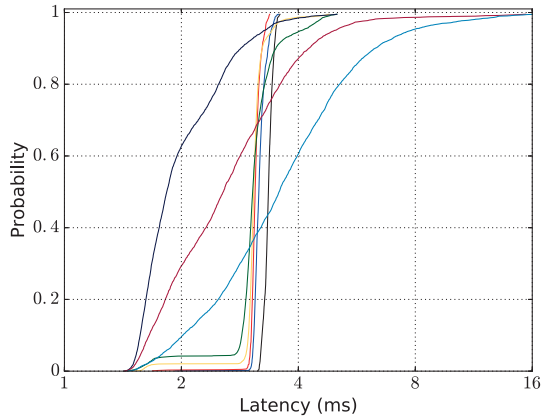
(b) i5-4570



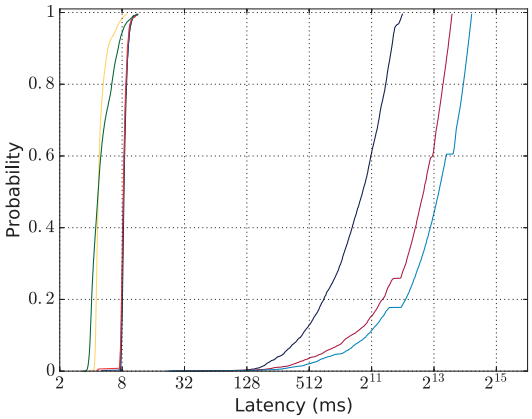
(c) Q9650



(d) VPS-Xeon

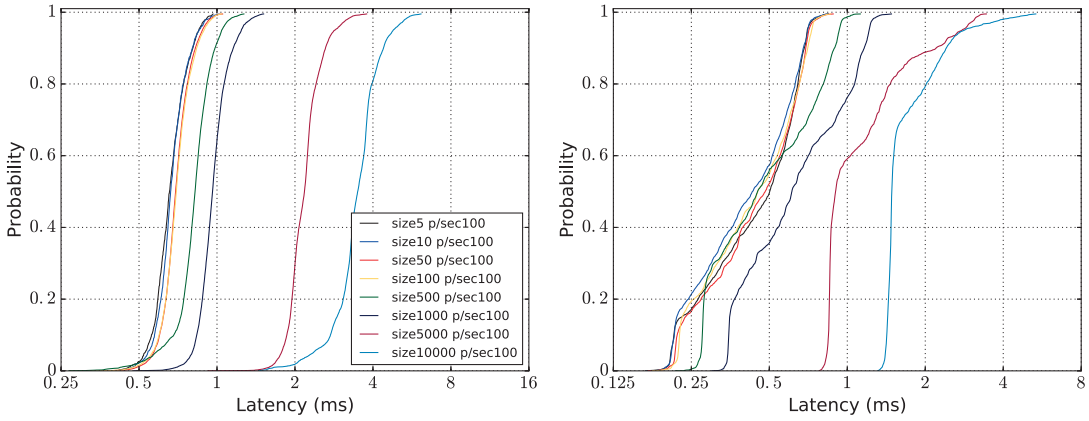


(e) Rasp3



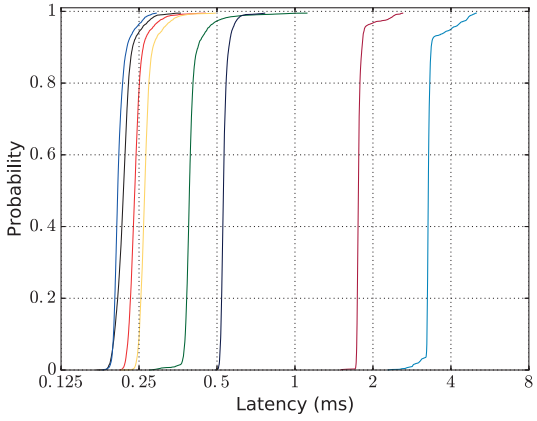
(f) BBB

Figure 4.21 – CDF of latency introduced by vNetwork hosted on varying hardware, for varying packet/sec and 500 B payload

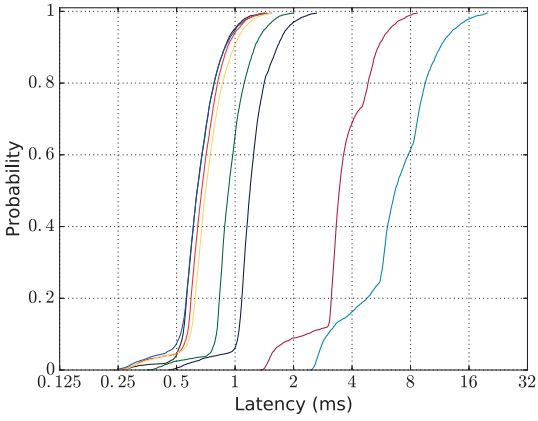


(a) i7-6700

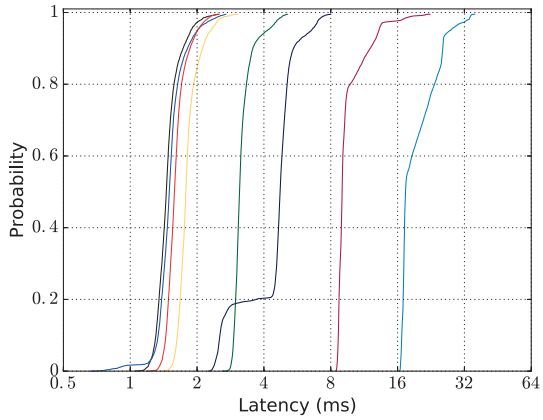
(b) i5-4570



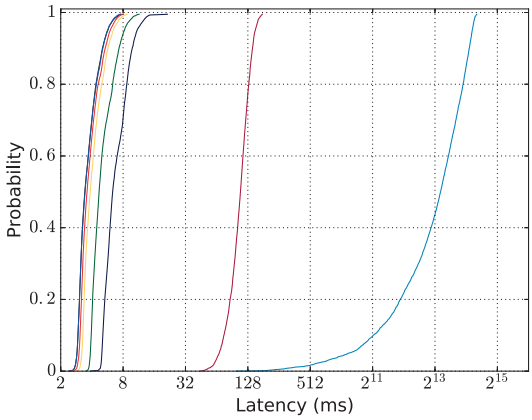
(c) Q9650



(d) VPS-Xeon

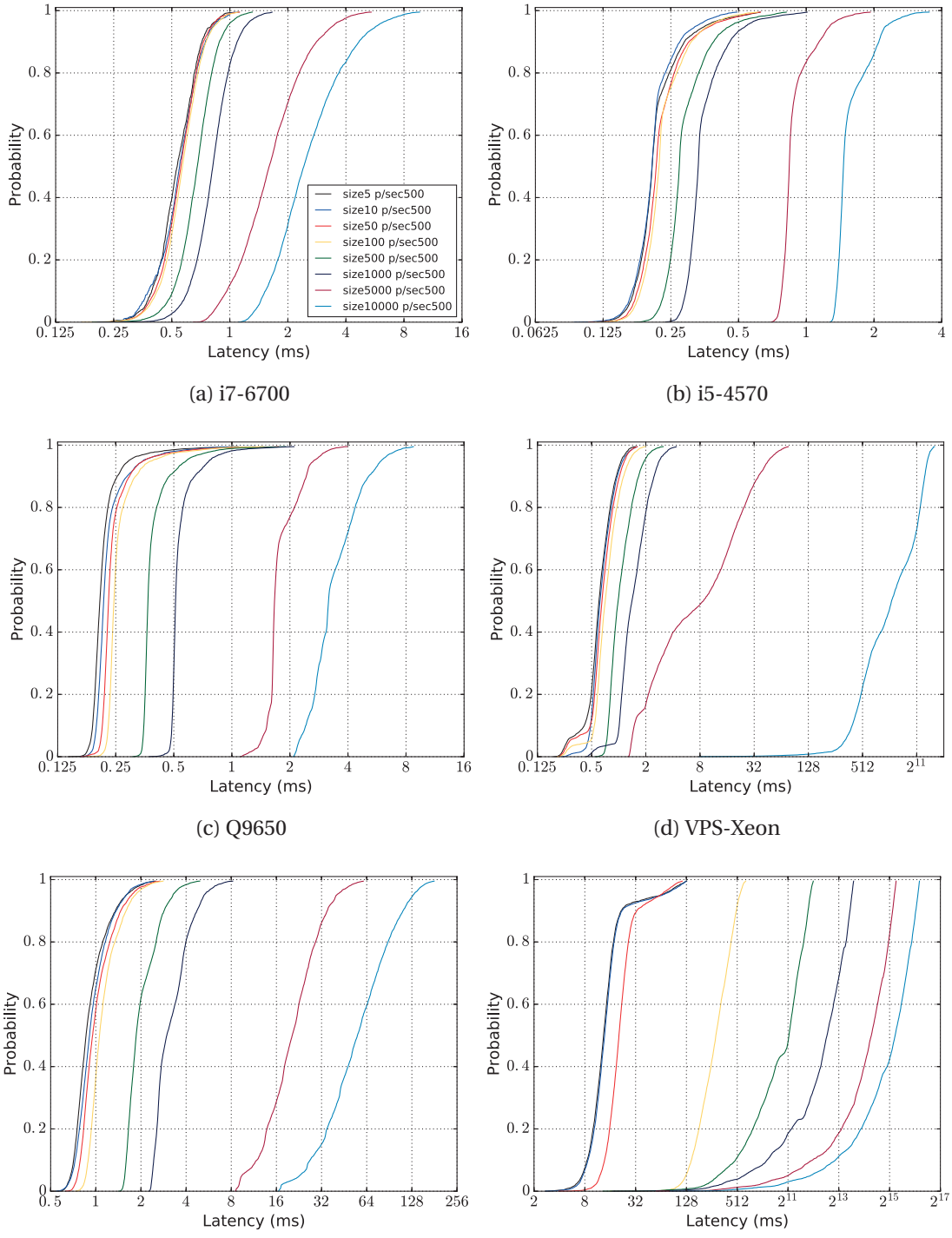


(e) Rasp3



(f) BBB

Figure 4.22 – CDF of latency introduced by vNetwork hosted on varying hardware, for 100 packet/sec and varying payload size



(a) i7-6700

(b) i5-4570

(c) Q9650

(d) VPS-Xeon

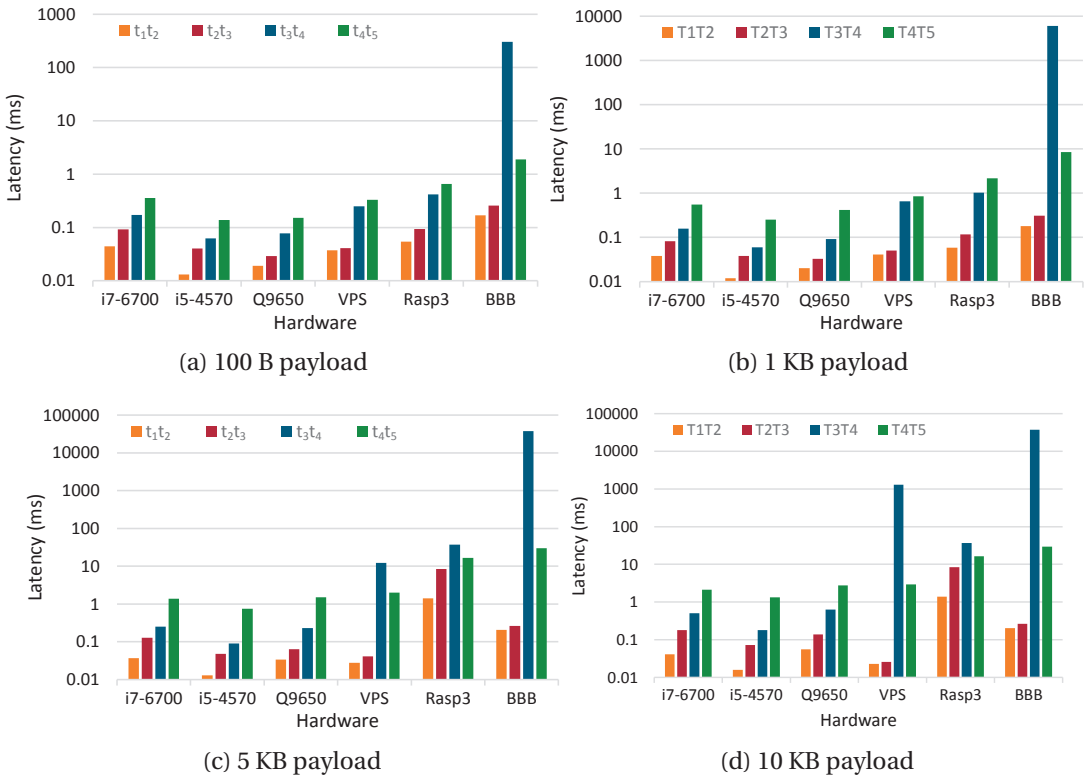
(e) Rasp3

(f) BBB

Figure 4.23 – CDF of latency introduced by vNetwork hosted on varying hardware, for 500 packet/sec and varying payload size

Table 4.2 – Mean and standard deviation of total added latency ($mean \pm std$) in ms by the vNetwork stage for various tests and hardware.

Configuration		Hardware					
size	**freq**	**i7-6700**	**i5-4570**	**Q9650**	**VPS**	**Rasp3**	**BBB**
100 B	**1 p/s**	1.02 ± 0.12	0.72 ± 0.13	0.27 ± 0.03	1.18 ± 0.20	2.13 ± 0.12	5.87 ± 2.06
100 B	**5 p/s**	1.02 ± 0.12	0.66 ± 0.13	0.26 ± 0.02	1.41 ± 0.18	2.04 ± 0.14	5.75 ± 1.48
100 B	**10 p/s**	1.03 ± 0.11	0.55 ± 0.20	0.25 ± 0.02	1.00 ± 0.38	2.02 ± 0.15	5.68 ± 1.64
100 B	**50 p/s**	1.02 ± 0.12	0.52 ± 0.18	0.25 ± 0.07	0.76 ± 0.27	1.83 ± 0.25	3.71 ± 1.41
100 B	**100 p/s**	1.01 ± 0.13	0.46 ± 0.18	0.24 ± 0.09	0.66 ± 0.21	1.71 ± 0.46	4.21 ± 1.82
100 B	**500 p/s**	0.66 ± 0.18	0.25 ± 0.19	0.27 ± 0.16	0.70 ± 0.34	1.10 ± 0.41	38.2 ± 58.6
100 B	**1k p/s**	0.38 ± 0.12	0.19 ± 0.06	0.32 ± 0.37	0.96 ± 0.65	1.47 ± 0.65	$4.4k \pm 2.6k$
100 B	**2.5k p/s**	0.20 ± 0.07	0.16 ± 0.10	0.39 ± 0.41	2.80 ± 3.18	1.88 ± 1.09	$7.1k \pm 4.3k$
500 B	**1 p/s**	1.19 ± 0.13	0.71 ± 0.26	0.39 ± 0.03	1.44 ± 0.30	3.30 ± 0.25	7.87 ± 2.96
500 B	**5 p/s**	1.20 ± 0.12	0.86 ± 0.13	0.37 ± 0.02	1.38 ± 0.24	3.13 ± 0.28	7.71 ± 2.15
500 B	**10 p/s**	1.22 ± 0.12	0.85 ± 0.14	0.37 ± 0.02	1.51 ± 0.29	3.10 ± 0.22	7.62 ± 2.29
500 B	**50 p/s**	1.20 ± 0.14	0.74 ± 0.20	0.35 ± 0.03	1.21 ± 0.30	3.07 ± 0.28	4.79 ± 1.38
500 B	**100 p/s**	1.20 ± 0.16	0.52 ± 0.24	0.37 ± 0.12	0.98 ± 0.40	3.21 ± 0.62	5.80 ± 3.12
500 B	**500 p/s**	0.69 ± 0.16	0.29 ± 0.09	0.40 ± 0.28	1.01 ± 0.50	1.98 ± 0.71	925 ± 575
500 B	**1k p/s**	0.51 ± 0.17	0.24 ± 0.05	0.48 ± 0.56	1.74 ± 0.92	2.90 ± 2.28	$6.4k \pm 3.5k$
500 B	**2.5k p/s**	0.29 ± 0.08	0.22 ± 0.07	0.57 ± 0.68	30.3 ± 57.7	4.06 ± 2.28	$6.9k \pm 5.5k$
5 B	**100 p/s**	0.66 ± 0.09	0.46 ± 0.17	0.22 ± 0.02	0.55 ± 0.22	1.58 ± 0.28	3.81 ± 1.65
10 B	**100 p/s**	0.67 ± 0.09	0.44 ± 0.17	0.21 ± 0.01	0.49 ± 0.18	1.61 ± 0.26	3.87 ± 1.63
50 B	**100 p/s**	0.70 ± 0.09	0.47 ± 0.17	0.24 ± 0.03	0.67 ± 0.38	1.78 ± 0.25	4.07 ± 1.77
1 KB	**100 p/s**	0.98 ± 0.14	0.69 ± 0.31	0.53 ± 0.04	1.18 ± 0.40	4.50 ± 1.15	7.79 ± 3.86
5 KB	**100 p/s**	2.20 ± 0.35	1.22 ± 0.60	1.78 ± 0.12	3.80 ± 1.32	9.69 ± 2.04	108 ± 25.3
10 KB	**100 p/s**	3.53 ± 0.72	1.75 ± 0.63	3.33 ± 0.31	7.34 ± 3.33	19.8 ± 4.25	$9.5k \pm 5.5k$
5 B	**500 p/s**	0.55 ± 0.13	0.22 ± 0.07	0.23 ± 0.17	0.63 ± 0.16	1.10 ± 0.39	8.48 ± 0.56
10 B	**500 p/s**	0.56 ± 0.14	0.21 ± 0.05	0.24 ± 0.19	0.62 ± 0.18	0.95 ± 0.29	8.31 ± 0.66
50 B	**500 p/s**	0.56 ± 0.13	0.23 ± 0.07	0.26 ± 0.22	0.63 ± 0.39	1.09 ± 0.35	24.4 ± 16.7
1 KB	**500 p/s**	0.83 ± 0.20	0.36 ± 0.10	0.56 ± 0.23	1.51 ± 0.71	3.36 ± 1.09	$6.0k \pm 3.5k$
5 KB	**500 p/s**	1.79 ± 0.83	0.90 ± 0.19	1.83 ± 0.50	14.2 ± 15.3	22.2 ± 9.84	$20k \pm 11k$
10 KB	**500 p/s**	2.85 ± 1.48	1.60 ± 0.36	3.57 ± 1.22	$1.3k \pm 837$	63.3 ± 33.6	$37k \pm 21k$



(a) 100 B payload

(b) 1 KB payload

(c) 5 KB payload

(d) 10 KB payload

Figure 4.24 – Average latency for each vNetwork pipeline stage, hosted on varying hardware, for 500 packet/sec and varying payload size

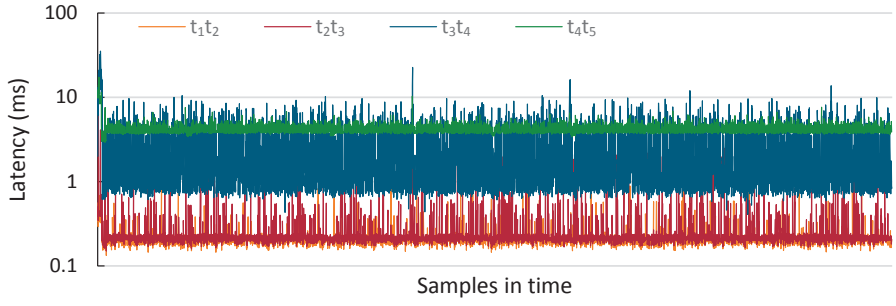
As mentioned in subsection 4.5.5 the vNetwork consists of various stages in an asynchronous pipeline. When the flow of packets through it is consistent, each stage execution introduces comparable latency. On the contrary, due to different algorithm complexity in each stage, in high load situations, not all of the stages are stressed equally. Hence, their contribution in overall latency does not increase linearly with the load.

Fig. 4.24 proves this fact on different hardware for 500 packets/sec of varying payload size. The $t_n t_{n-1}$ designators refer to the time difference $t_{n-1} - t_n$ where t_i the timestamp at step i of the pipeline as seen in the figure 4.14. As expected, the powerful machines are not affected by the increased payload size. However, the VPS, Raspberry, and BeagleBone show significant delays, mainly on the $t_3 t_4$ time frame. This corresponds to the Twisted thread which cannot keep up with the increasing load. However, on the date of this writing, the exact cause of this accumulating delay on the Twisted reactor is still unclear. A valid assumption is that the microprocessor units are unable to handle the computational complexity of the Twisted library in high load scenarios. An additional interesting observation is that for the Raspberry Pi the Twisted latency is increasing in much smaller rate compared to the other two.

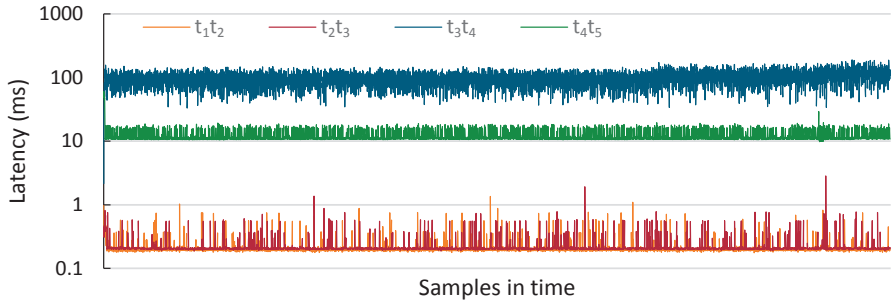
Final analysis for the vNetwork module is visible in Fig. 4.25. The three figures plot the samples of the latencies over time for three payload sizes while running on the BeagleBone. The plotting versus time enables the observation of the transient phenomena in the network simulation

module. The $t_3 t_4$ time frame continues to be of interest. Although in Fig. 4.25(a) and 4.25(b) the added latency is bounded; in Fig. 4.25(c) of the largest payload, a latency accumulating effect is observed. Thus, the Twisted reactor accumulated messages in the input faster than it can push them.

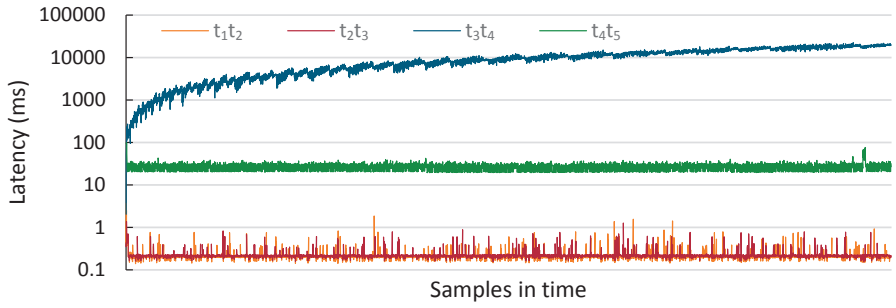
Concluding, the vNetwork in conventional and not benchmarking loads is highly efficient even on low-power hardware such as the BeagleBone. Additionally, if need be, thanks to the modular and socket based design, multiple processes can be launched, better leveraging multi-core architectures or even distributed nodes topologies. A load balancer can allocate the packets accordingly enabling both horizontal and vertical scaling.



(a) 1 KB payload



(b) 5 KB payload



(c) 10 KB payload

Figure 4.25 – Latency over time for each vNetwork stage, running on the BeagleBone, for 100 packet/sec and varying payload size

4.6.4 vBuilding: emulator practical assessment

The vEngine allows an existing BMS to collect data from the virtual infrastructure much like its physical counterpart. In reverse, the BMS can transfer downstream commands to the virtual middleware, emulating their control. This creates the opportunity for the energy management system (EMS) to develop energy management strategies by integrating both physical and virtual future devices in the actual building. Enabled by the BMS services, the EMS remains agnostic to the underlying components and communication protocols while it can experiment with non-existing, virtual infrastructure.

In order to evaluate the practical functionality of the vEngine, a virtual building has been created with the elements listed in Table 4.3. The *vUser* triggers the activities of nearly all the loads during the day based on its model. The EMS on the other hand controls the *vHeater*, *vBattery* and *vWasher* while efficiently managing the energy.

Table 4.3 – List of emulated elements in the virtual building and their consumption features

		Power range	
Name	**Element**	**Regular use**	**Peak use**
vUser	Occupant	-	-
vBase	Baseline consumption	[220, 240]	[340, 350]
vComputer1	Computer 1	[50, 80]	[110, 130]
vComputer2	Computer 2	[50, 80]	[110, 130]
vLight	Dimmable Ceiling lights	[10, 100]	[100, 100]
vHeater	Heating system	[50, 600]	[600, 600]
vWasher	Washing machine	[450, 650]	[800, 1200]
vBattery	Storage battery	[0, 330]	[330, 330]

Fig. 4.26 illustrates the virtual building powers throughout the day. It highlights the high potential the engine as an auxiliary tool for simulating a building in real time and improving the EMS. Easy reconfigurability of the vEngine, allows the EMS to test many scenarios and configurations without costly investments.

The resulting flexibility is leveraged by the EMS for meeting the energy objectives, reducing energy bill and providing ancillary services to the grid. Fig. 4.27 illustrates, for example, the reduction of the power peak demand with the help of the EMS. The first column of figures denotes the original consumption profile. In the second column the EMS performs a load shifting by postponing the *vWasher* cycle about 5 hours later which results in much smoother building power profile. If the virtual building also supports battery storage like in the third column, the EMS mitigates the peak by introducing energy from the battery without rescheduling the *vWasher*. While both techniques can be used in traditional demand response (DR) scenarios, the research is now progressively moving towards real-time pricing (RTP) and

critical peak pricing (CPP) programs [234, 235, 236]. Those pilot programs demonstrated that consumers adjust their electricity usage in response to price changes [237]. The EMS could facilitate the process and increase the financial gains by charging the battery low price periods and to discharging it when it's financially profitable. Hence, the building emulator and the *vBattery* in particular, can be used as practical and realistic financial gains simulation tools and research-supporting technologies for improving the EMS algorithms.

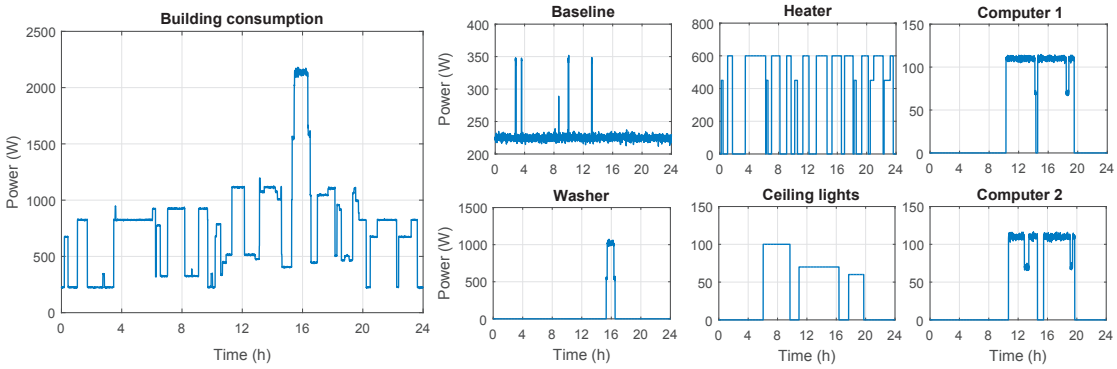


Figure 4.26 – Consumption profile of building and its virtual elements

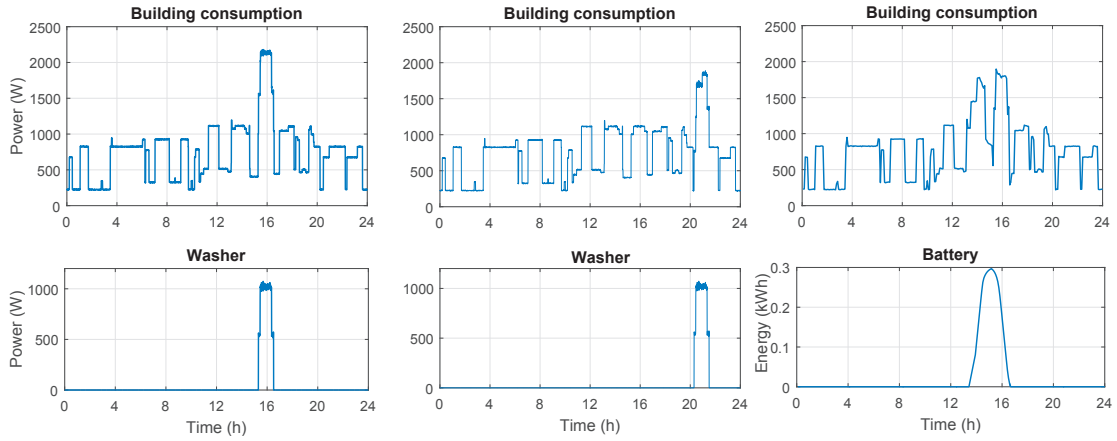


Figure 4.27 – Consumption profile of building with enabled peak power reduction

Moreover, the EMS may leverage the virtual infrastructure for evaluating the energy optimization potential within physical buildings, actual occupants and real infrastructure. Using the virtual energy-flexible entities (e.g. battery, PV panel), the EMS can provide energy and financial performance estimates justifying or not a candidate retrofitting investment.

To evaluate this closely integrated cyber-physical functionality of the vEngine in the context of the EMS, an actual university campus building has been augmented with the vEngine. The virtual elements complement the already installed sensor and actuator network. Those virtual devices are a *vPVpanel* and a *vBattery* which are enough for testing simplified peak power mitigation techniques.

Since the *vPVpanel* is using the same model described in subsection 4.5.3, it gathers the outside luminosity and temperature in real time and emulates the appropriate power production for each specific day. An alternative approach would have been to create two *vSensors* with values extracted from the *Measurement and Instrumentation Data Center*'s online databases. The model of *vPVpanel* is using the parameters of Solar's DIAMOND CS6X-310P cell. The *vBattery* is also using the already introduced model with simplified parameters: $\{Capacity: 3kWh; Power: 3.3kW; efficiency: 95\%\}$.

The EMS controls the storage system in order to mitigate the peak power and flatten the profile of the power purchased from the SG, using the logic of Eq. 4.27, where P_B is the power of the battery, P_L is the building consumption, P_G the generated power and $\{P_{th}^{min}; P_{th}^{max}\}$ are set to $\{1.5; 2.2\}kW$.

$$P_B = \begin{cases} P_{th}^{min} - P_L + P_G & \text{if } (P_L - P_G) < P_{th}^{min} \\ P_{th}^{max} - P_L + P_G & \text{if } (P_L - P_G) > P_{th}^{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.27}$$

Finally, Fig. 4.28 shows a realistic demonstration of EMS capabilities using virtual generation and storage and physical power sensors as recorded on the 3^{rd} of August 2016. The load consumption is, in fact, the real measurement as recorded by the smart meter. The simple EMS intelligence manages the battery in a way that a reduced amount of energy is purchased from the network.

Fig. 4.29 visualizes the use of such building-in-the-loop tool for investigating potential investment. The area between the two graphs denotes the reduction of purchased energy in a single day. A payback period analysis, cf. Eq. 4.28, or even better a discounted payback period can then estimate the number of days required to recover the cost of the investment in storage and generation. At the end of this period, the investor not only will reduce the carbon footprint but also start to make money from the electric bill. Therefore, despite this been an oversimplified financial analysis of the investment, the benefits remain clear. The critical data to be extracted accurately is the emulated reduction in energy as a result of the virtual generation and/or storage. The rest is a matter of the used financial model for calculating the investment potential.

$$Payback\ Period = \frac{Cost\ of\ investment}{Reduction\ in\ energy\ bill} \tag{4.28}$$

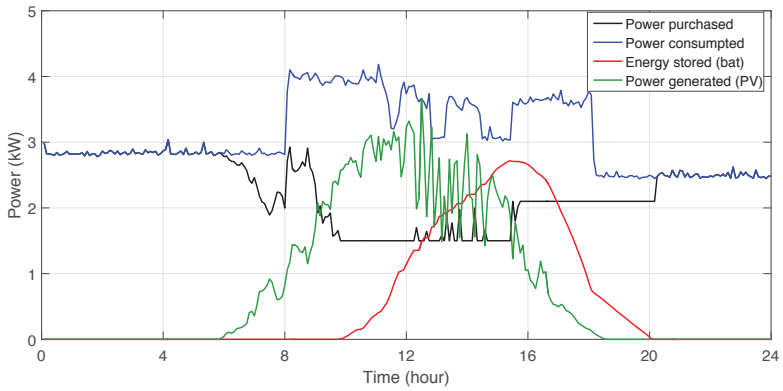


Figure 4.28 – Practical demonstration of EMS capabilities using virtual generation and storage

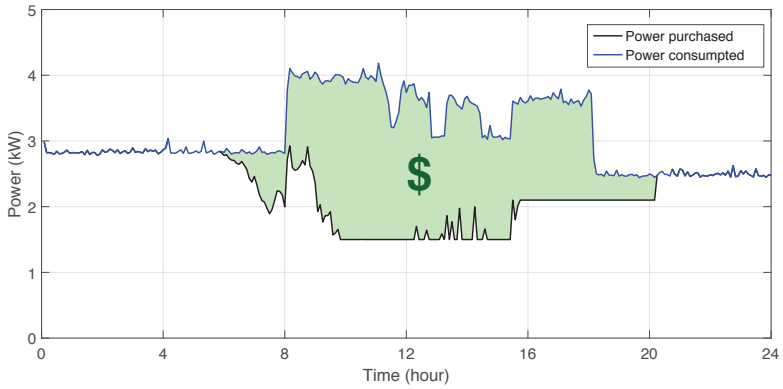


Figure 4.29 – Financial benefits predicted by the EMS for real consumption using the virtual generation and storage

4.7 Conclusions

The SB is more than ever in a transitional phase due to the introduction of IoT, novel energy storage technologies, improved energy generation infrastructure and highly intelligent algorithms. Nevertheless, all these technologies will not be ready or financial advantageous at the early stages.

This chapter presented an innovative SB emulation system for integration in a BMS. The emulator can be used either in real-time along with existing infrastructure, or offline as a software simulator tool. The potential of this tool for supporting SB energy management research has been presented. Moreover, the practical application of financial analysis on retrofitting investments has also been justified. In fact, such tool could mitigate the burden of retrofitting, and the risk of fragmented IoT market by easing the adoption of new technologies.

The DES theory and the unique design formulation developed in this work, enables the implementation of a lightweight, highly concurrent general purpose emulation engine. It is proven to offer high performance on various hosting hardware, including embedded platforms.

The system can accurately emulate several different SB elements including its occupants and embedded networks based on literature-proved models. Its software design is not only scalable on distributed infrastructure but also expandable regarding supported emulation models.

The emulator can be used either in real-time along with existing infrastructure, or offline as a software simulator tool. The potential of this mechanism for supporting SB energy management research has been presented. Moreover, the practical application of financial analysis on retrofitting investments has also been justified. In fact, such tool could mitigate the burden of retrofitting, and the risk of fragmented IoT market by easing the adoption of new technologies.

5 Smart Building Case Study

The proposed architecture has been validated in the previous chapters on a per subsystem basis. However, the analysis and the scope of a systems-thinking dissertation would not have been complete without a holistic system evaluation. This short chapter serves that purpose, and presents a realistic case study of the proposed Smart Building (SB) system architecture. The application of the case study is on a university campus building in which the proposed system has been installed. Firstly, it effectively demonstrates how some commercial and experimental information and communication technology (ICT) devices can be interconnected and leveraged for their sensing and actuation abilities. Secondly, it showcases the deployment and the resulting advantages of the distributed middleware architecture in this building construction. Finally, the chapter illustrates how an energy management system (EMS) is leveraging the exposed SB system services for monitoring and control of the building, in order to enable the demand side management (DSM) strategies.

5.1 Introduction

Previous chapters explored the various software and hardware architectures which were segmented, designed, and implemented as part of the complete Smart Building (SB) system architecture. Each chapter included a narrow-scope validation either through targeted case studies or via a performance evaluation procedure. This chapter aspires to provide closure to this heterogeneous design endeavor. It presents a holistic case study on the potential of the proposed deployed in a physical building for intelligent energy management and Smart Grid (SG) integration.

The case study offers a practical illustration of a building retrofitting using the proposed architecture, coupled with a selected number of information and communication technology (ICT) devices. Moreover, such a case study highlights how the energy management system (EMS), which is currently under development in the research group, connects to such a system and how it can leverage the system for providing ancillary services to the SG through demand response (DR) schemes.

The research study of this dissertation was part of a broader scope "Smart Grid Campus" project in École polytechnique fédérale de Lausanne (EPFL). The goal of this project was distribution energy grid modernization by monitoring in real time the consumption, production, and storage for several buildings on the campus. Fig. 5.1 illustrates this project. For most buildings, the distributed phasor measurement unit (PMU) monitoring systems collect and time align asynchronous power data. Those are then transmitted to a centralized control server with minimum latency and high security. However, for one of those locations, c.f. Fig 5.1 *2.ELB*, an alternative approach was followed. Instead of a PMU monitoring with aggregated power data, the concept of SB has been adopted. The vision behind this decision was the evaluation of the potential of the SB as an interactive participant in the coordination of the distribution energy network.

5.2 Physical Building and its Challenges

The selected building on the university campus where the system has been deployed is a 4-floor construction. The second floor has been selected as the main target for retrofitting. Fig. 5.2 illustrates the floor plan. As with many office-oriented constructions, there are several similar rooms adjacent to each other, and some common larger spaces used as laboratories and meeting places. Despite the second floor being the prime target for the system installation, for simplicity's sake and without loss of generality, this chapter extends the analysis to the complete building.

This campus building is an excellent representation of an office-commercial building in a rather old construction. Moreover, the occupants' patterns are less stochastic, as they follow the usual work and classroom hours. Thus, it is an ideal candidate for retrofitting not only for automation, but also for evaluation of more advanced ambient intelligence and energy

Figure 5.1 – The EPFL "Smart Grid Campus Project", in green the smart building and orange the PMU locations

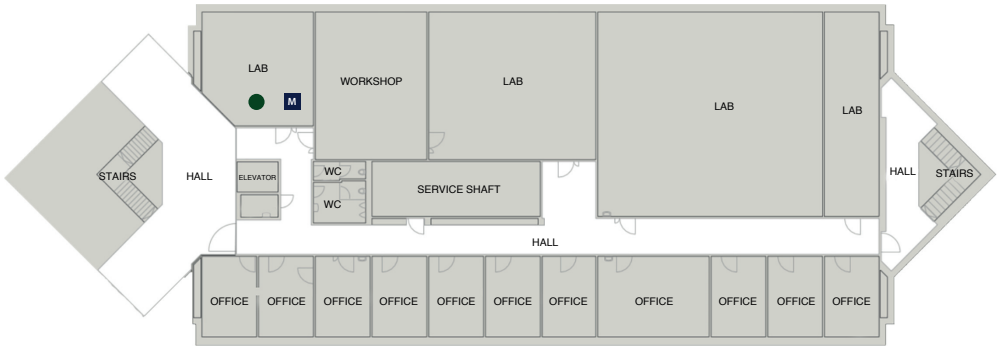


Figure 5.2 – EPFL ELB building's 2nd floor plan

management systems in a more deterministic environment. However, the conversion of the particular building construction to a "smart" one has revealed numerous challenges along the way, which are due to the particular building architecture, the selected ICT devices, and the internal existing cabling design. The main challenges are listed below.

- As is easily observable in Fig. 5.2, the building is disproportionately long in one dimension. It requires similarly proportioned ICT networks in order to completely monitor the internal living spaces. However, retrofitting additional cables solely for the purpose of sensor communication was not an option. Thus, many building automation standards that require this physical medium of communication have been excluded. The alternative viable option was to use wireless communication standards. However, unless the point-to-point range of such sensors is long enough to for a star topology to cover the whole floor, a mesh topology is a more appropriate choice. Still,

communication latency can become significant if a message between two communicating sensors needs to be relayed by several nodes in between. If one also considers the low power operation of such relays, the latency can be detrimental to the real-time management of the building. Moreover, for battery-powered relay nodes, this would also create a considerable impact on their autonomy, due to their relaying activity.

- Due to floor construction materials, wireless performance across floors has been found to be unreliable and low performing for building management.
- Alternatively, the use of power line communication (PLC) for load monitoring and control was also tested. However, the building has a peculiar electrical cabling for the three power phases; for example, a single room may have up to three different phases behind the electrical outlets. Thus, PLC-enabled devices could not communicate within the same room without additional phase bridging electronics.
- Furthermore, the high population of always connected loads with switching power supplies (e.g., computers, servers, screens, etc.), severely impacts the physical performance of the PLC physical medium. The reasons for such performance degradation, while shortly investigated, are beyond the scope of this dissertation.
- Finally, as with most large office buildings, there is a high density of loads, occupants, and their activities. This generates an enormous number of events and data by the ICT that need proper management with the least possible latency.

Those challenges are equally significant for both retrofitting with management technologies and for efficiently operating them throughout the lifecycle of the building. The alternative, legacy building automation system (BAS) based solutions, would need considerable workhours for adaptation to this particular building. This process significantly increases the overall cost and possibly jeopardizes the performance and reliability of the system. It is in this non-ideal building setup that the proposed system architecture of this dissertation shows significant advantages. The purpose of the following sections is to highlight such benefits in a realistic environment of a practical SB retrofitting.

5.3 System Deployment

The proposed system design would vary depending on the stakeholders' particular priorities in each subsystem. Nevertheless, the primary system architecture remains the same as seen in Fig. 5.3.

The EMS scrutinized in the following Section 5.4, acts as the interface between the SB and the SG. It is essentially the SG stakeholder in the building as it provides to the former ancillary services through an array of DR scenarios. As the building management system (BMS) provides a web and real-time application programming interface (API), the EMS can be a local or a remote software system. For the purpose of this case study, the EMS is hosted on the local computer of the researcher developing it.

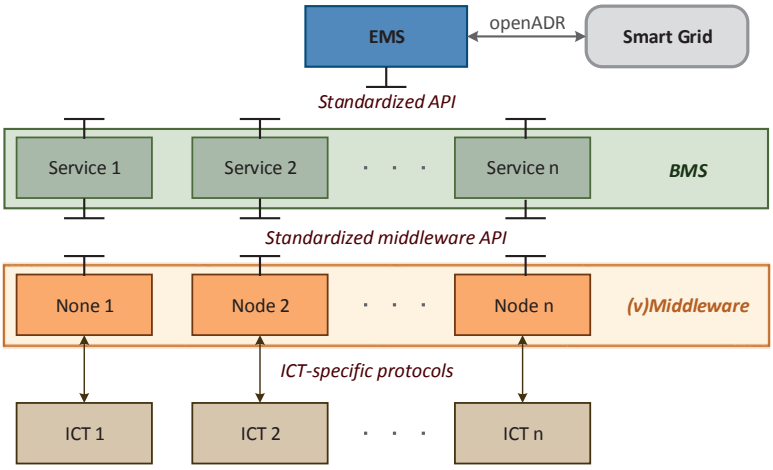


Figure 5.3 – Holistic system architecture

On the other hand, the BMS application and real-time server are both hosted on the cloud, on a single virtual private server (VPS). A second VPS hosts the time series database (TSDB). A single, non-distributed BMS is preferred, given the size of the building and the activities within. In the case of increased future computational demand, the VPS can easily scale to accommodate the demand. Concerning the modeling aspect as presented in Chapter 2, the whole building is represented by a `Unit` model, and each room, naturally, with a `Room` model. The thermal and localization modules, used for the validation section of the same chapter, provide high-level data, critical for optimal EMS operation. Thus, when available, wall and window material models are enabled for the thermal simulator models. Finally, a fallback BMS server is provisioned on the building premises which runs on a single Raspberry Pi 3 hardware in case there are connectivity problems with the cloud-hosted BMS.

As the overall project seeks to modernize power distribution across the university campus, high granularity of load measurement capability within the building is of paramount importance. However, the available ICT systems for sensing/monitoring and actuation/controlling capabilities of the building loads are vast and heterogeneous. Nonetheless, as most of them serve to share the same scope, for this specific case study, some ICT systems were subjectively selected in order to meet most of the application requirements.

To monitor and control the appliances of the building, the ICT device designed and built by eSMART, a Swiss company, was selected. As seen in Fig. 5.4, those modules can monitor the power of two loads at the same time. They also offer a means of actuation over the loads with dimming and switching capabilities. However, the aspect that sets them apart from the competition is their distributed communication through the building powerlines. This eliminates the inconvenience of any additional wiring infrastructure installation, reducing the overall cost and time for retrofitting. Moreover, the lack of communication over the air mitigates the ISM band congestion, an important aspect considering the amount of continuously communicating Internet of Things (IoT) devices to exist in each SB. Besides

power measuring, their favorable size, thanks to microelectronics innovations, allows this miniature device to be installed behind regular wall sockets. Therefore, they are completely transparent to the occupant, having zero visual impact on the architecture and interior design of the building.

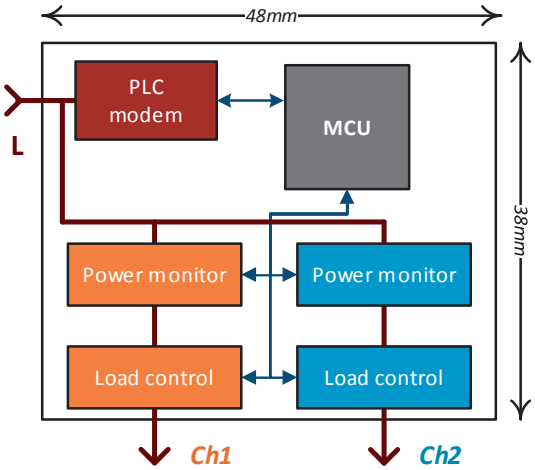


Figure 5.4 – The eSMART power monitor and load control module

For the environmental monitoring data, a 6LoWPAN-capable, PV energy harvesting multi-sensor developed within the scope of the overall project was used. Those multi-sensors were introduced in key locations in the building, where there was adequate solar energy for powering them. Fig. 5.5 illustrates this: on the left is the block schematic diagram of this multi-sensor, and on the right is the 3D model of the final product. The device embeds the following sensors: temperature, humidity, pressure, luminance, air quality, motion (PIR) and a microphone. The multi-sensor system is built around an ARM® Cortex-M3 with integrated IEEE 802.14.5 2.4Ghz transceiver as a system on a chip (SOC). Except for the microphone and the PIR, all sensors communicate through an I2C interface to the SOC. Through the I2C power switch, the SOC directly controls the power supply lines of each sensor interface, eliminating any power drain, even quiescent current, when they are not necessary, improving the overall autonomy. Finally, a USB connection is available for quick battery charging and configuration via a serial terminal. The power measurement unit (PMU) harvests with nearly 80% efficiency the energy from the PV panel to recharge the battery and power the system. Finally, to actively manage the energy budget, a voltage and current measurement chip is embedded, which measures both solar and battery input power.

Concerning the data modeling and representation of those sensors and actuators within the BMS, the recommendations of Chapter 2 were followed. Each eSMART device is modeled as two `Sensor` and two `Actuator` models, one for each module's channel. The energy harvesting multi-sensor modules were also modeled as a collection of `Sensor` models, one for each physical or virtual sensor interface, including for example the internal battery level. Nonetheless, each model instance has a primary unique ID and shares the same secondary ID which identifies the specific physical ICT instance. Based on what the hardware could control

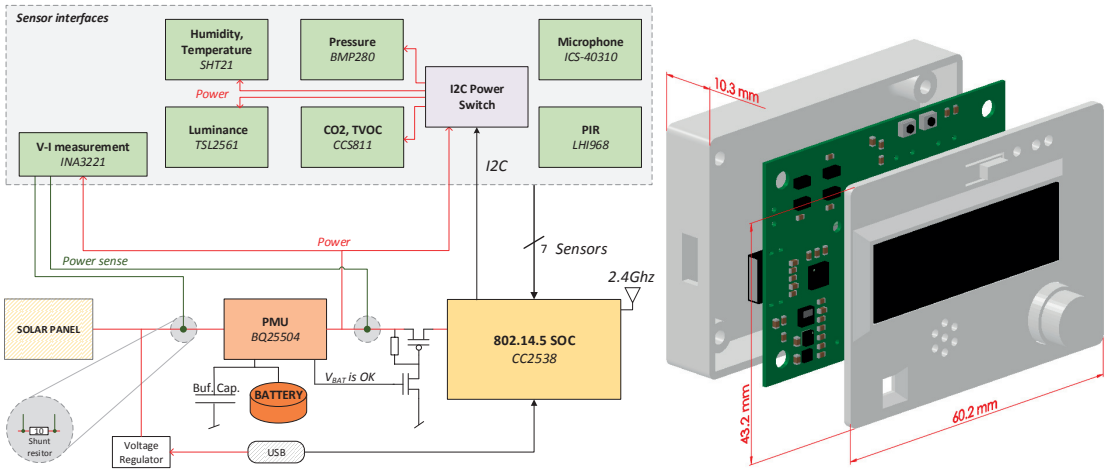


Figure 5.5 – The 6LoWPAN-enabled and PV energy harvesting environmental multi-sensor

and measure, the model instances were configured accordingly for exposing the semantic abstractions through the BMS API.

As already mentioned in the relevant chapter, the middleware is not only providing abstraction to the ICT standards, but also low-level connectivity where the embedded networks reach their performance and range limits. Practical tests have proven that a single middleware node is not enough to cover the whole floor. For this reason, five middleware nodes have been provisioned and placed across each floor according to the connectivity needs. Fig. 5.6 illustrates the middleware topology for a single floor, and how it interconnects the various embedded networks. In the figure, with dark blue are illustrated the middleware nodes and their connections. Additionally, each electrical phase gets its own color and the ICT for load measurement and control are labeled with "L". Finally, the green cycles with the "S" letter, denote the wireless environmental sensors. The following paragraph scrutinizes how the middleware overcame the deployed ICT devices' limitations.

Firstly, in order to overcome the inherent limitations of the PLC physical medium, a couple of middleware nodes are interfacing with the different power phases of the building. Thus, even if the PLC modules are unable to communicate over different power phases, they are interconnected and abstracted over the middleware layer. Moreover, in order for the energy harvester sensors to remain battery efficient, a star network topology was adopted. In this case, the middleware nodes are acting as a border router to the 6LoWPAN network and enable their connectivity over the middleware layer. Regardless of the interfaced embedded network, all of the nodes' software is running on the tested LinkIt Smart 7688 hardware and communicate using the onboard Ethernet interface and cabling of the building. Lastly, to improve the security, as scrutinized in Chapter 3, each middleware node communicates within a virtual private network (VPN) tunnel with its peers and the BMS.

Finally, due to the age of the construction, the building lacks any generation and storage

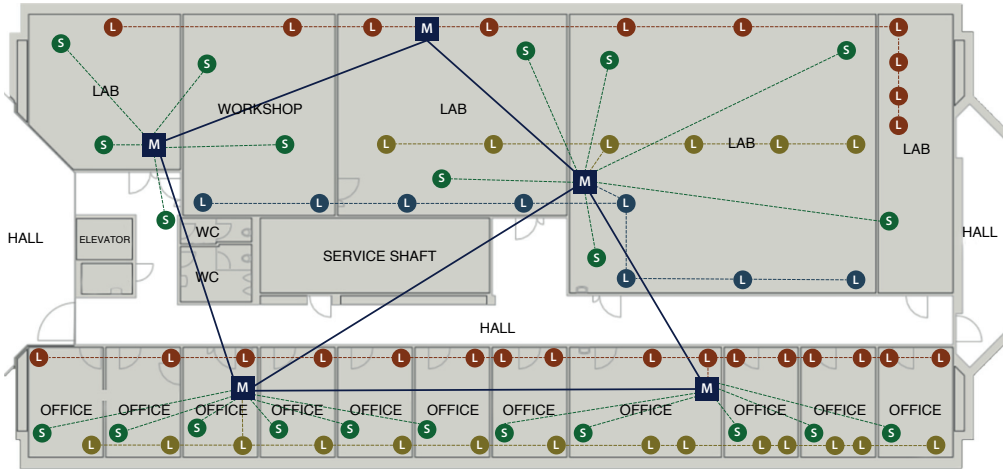


Figure 5.6 – The middleware and embedded networks topology on the actual building

infrastructure. However, the EMS-oriented studies in the research group require such capabilities. It is therefore necessary to have such services exposed by the SB management system in order to test and validate the energy optimization algorithms. The "building-in-the-loop" system, as introduced in Chapter 4, is a viable solution. Therefore, thanks to the real-time sensor data and this virtualization engine, the generation and storage capacity of the building can be emulated in parallel with the real environmental, occupancy, and load patterns. Based on this array of heterogeneous data, the EMS can interact with the physical load, the virtual storage and generation, as well as the SG, for applying the appropriate energy strategies.

5.4 Energy Management System

The EMS, like any other intelligence algorithm, connects on top of the BMS using the extensive API. This enables the interaction of any external systems with the semantically exposed elements of the building as modeled in Chapter 2. This section introduces the EMS and elaborates on the future architecture of the SB with enhanced energy management capabilities and demand side management (DSM) support. Fig. 5.7 illustrates such system architecture.

In Fig. 5.7, a red shape denotes the BMS core which integrates both the application and the real-time servers with the relevant data storage systems, cf. Chapter 2. At this level, the middleware provides the distributed connectivity abstraction to the monitoring and controlling infrastructure installed in the building. In the same figure, blue shapes denote the advanced intelligence modules that leverage the BMS core services. These modules are exposed as high-level data, compared to the primary ones of the BMS core, and they are essential for implementing the advanced energy management strategies.

To begin with, the *grid communication* modules handles, as the name suggests, the required

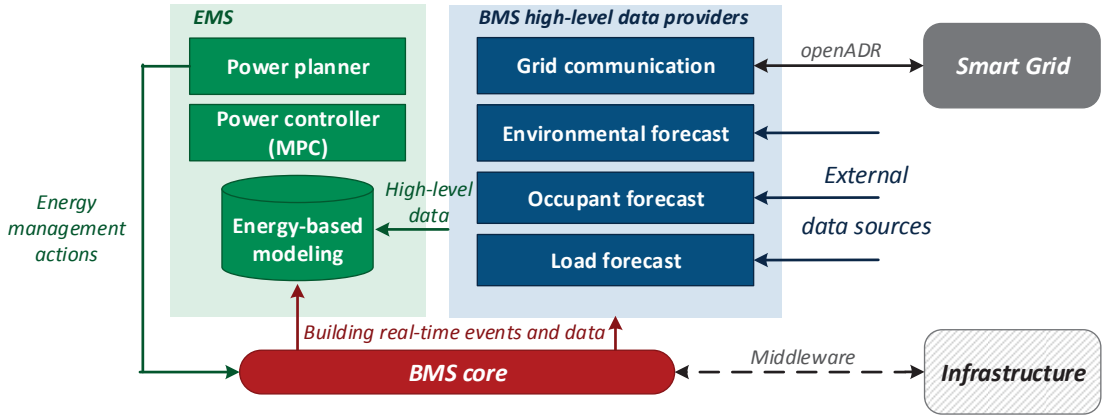


Figure 5.7 – System architecture for an energy management enabled building

knowledge exchange with the SG. The protocol selected for that purpose is the Open Automated Demand Response (OpenADR) [238]. The standard supports fully automated management of the load based on specific signals from an energy utility and provides direct connectivity to customers' energy management systems. In fact, OpenADR is a prominent standard and the foundation for DR interoperability. The *grid communication* module is stateful, and thus stores all the relevant information and communication states essential for an automated demand response (ADR) process to succeed. Those data are then fetched in real time by the EMS depending on the executing optimization scenario. Thus, the EMS remains not only agnostic to the building's ICT infrastructure, thanks to the BMS core, but it remains also generic regarding the DSM standard used by the energy utility of a particular building.

The *environmental forecast* module is generating insights on the building living environment based on external input, for example the weather forecast, and internal ICT sensed data. Those insights relate, for example, to the internal temperature or luminosity. On the other hand, the *occupant forecast* provides insights on the activities within the living spaces. It leverages the raw data inputs, such as the localization events, or some external sources, such as public transportation schedules, in order to generate the high-level analytics of current and future occupant activity. Similarly, the *load forecast* leverages the data from the previous two forecasters, in addition to the load monitoring data, and provides load power estimates either per load or for the whole building. This module is critical for the energy management and optimization functionality of the EMS.

At the time of this writing, the EMS is in its infancy. Therefore, the associated researchers expect extensive development, redefinition of the key modules, and even redirection of the core aims in the following years. Nevertheless, it is still worth it to introduce its early stage implementation and main functionality for the completeness of this case study. The EMS is illustrated in Fig. 5.7 in green.

The *energy-based modeling* is a secondary data model internal to the EMS. Such a data structure

is used by the EMS for its optimization process; it is a fusion of energy-relevant data from the BMS and additional ones generated by the EMS. It allows, for example but not limited to, the EMS to categorize certain loads according to their degree of controllability or to store the machine learning extracted load profiles. Examples of energy-relevant data retrieved from the BMS are:

- building geometry and rooms arrangement, e.g., locations, sizes, features, etc.;
- for each room: its sensors, actuators and load objects;
- for each load, the available data, e.g., mean power, profile, etc.;
- other energy related infrastructure models: building's generation, storage;
- occupants' data and their preferences.

The active management of the energy is divided mainly into two phases. In the planning phase, cf. Fig 5.7 *power planner*, the estimated data are used for optimization over a long-term horizon, generally over 24 hours. The aim of the planning phase is to minimize the overall energy cost while maintaining acceptable occupant comfort. Thanks, to this phase, the EMS can communicate to the SG the estimated building-wide consumption profile, for example over the following day, week or any other appropriate time frame. On the other hand, the *power controller MPC* corresponds to the online optimization phase, which manages in real time the power consumption, generation, and storage. To do so, it takes into account the event data from the building ICT, the *energy-based modeling* data, as well as any signals from the *grid communication* module.

In particular, the model predictive control (MPC) module optimizes an objective function by taking into account relevant models, forecasts, and existing constraints. A building thermal model, a solar generation model, or energy consumption profile are only some examples of such applicable models. The forecast data over a certain time horizon are supplied to the MPC module for generating corrective actions on the controllable elements of the SB. The constraints are taken into account in that process and reflect the building stakeholders' requirements. Finally, real-time events, either from the high-level data providers or the BMS, are used to trigger the matching MPC functionality.

5.5 Conclusions

In conclusion, the proposed system architecture adaptability and deployment have been demonstrated in a challenging university campus building. Moreover, the vision of utilizing the system for advanced energy management practices has been introduced. Finally, it was shown how the currently in development EMS enables active energy management in coordination with various SG policies.

6 Conclusions

The Smart Building (SB) will certainly be the next step in building evolution, improving value and long-term performance in resource usage and occupant satisfaction. However, at the time of this writing, the social, financial, and technological barriers are still hindering its widespread adoption. Moreover, the concept of "smart building" is frequently used interchangeably with "building automation" without a clear distinction between the two; this creates ambiguity and confusion in the market.

In order to overcome such challenges both in research and in the market, this dissertation approached the SB on a systemic basis. The output of such an endeavor is the research, design, implementation, and finally validation of a secure and scalable SB architecture that addresses several of the challenges and barriers identified in Chapter 1. That chapter assessed and defined the modern SB, highlighted its dissimilarity to contemporary building automation systems, and identified its major stakeholders and applications. Thus, the findings of that chapter have been the driver and motivation for this dissertation and the engineering implementation that followed.

The systems thinking required the segmentation of the holistic SB architecture into individually identified sub-systems; each one was addressed in their respective chapters. In fact, with the exception of the case study chapter, they can be considered as independent research studies on their respective domains. For each, the scientific approach has followed a similar pattern. Firstly, the chapter introduces the addressed challenges and justifies the research motivation. Then, following a detailed literature review and comparison with other designs, it suggests a novel architecture that extends the specific state of the art. The chapter continues with a detailed design and implementation, while an experimental setup, defines the assessment metrics and measures the performance for validation in accordance with the design requirements.

An architecture to reduce complexity, improve adaptability, and enhance the scalability of the SB is proposed in Chapter 2. The modeling of the building and the semantic abstraction not only enables the trivial expansion with new semantics, but also decouples the intelligence and

management software logic from information and communication technology (ICT). This portability and reutilization of the algorithms is a major step towards a shared ecosystem of algorithms and management software compatible with any SB, similar to the current state of mobile applications. Most importantly, the reduced complexity and improved extendability do not inhibit the scalability and adaptability of the system. A scalable and efficient architecture maintains the SB's performance and reliability regardless of the building's internal and external activities (event generators).

On multiple occasions, the literature has highlighted interoperability as a significant barrier to the adoption of smart infrastructure in general, and of SB in particular. This work has also identified physical construction diversity as a key challenge in the design of a universal SB with an attractive cost brought by the economies of scale. Those barriers were the motivation for the original research on a middleware solution for SB in Chapter 3. To the author's knowledge, this is the first time a distributed middleware for SB has been proposed, and one of the few to consider middleware as a solution to the interoperability challenge of embedded and ICT devices. The power of the proposed distributed middleware is its extendability with new standards and its adaptability to physically different buildings without hindering the expected behavior and performance of the SB system. A commonly practiced approach would have required several protocol translation gateways and a manual configuration in other to interconnect the heterogeneous devices and communication standards. On the contrary, the chapter validated how even a few low-cost and power embedded electronics can be leveraged as distributed middleware nodes within the building. In the end, such middleware acts as an ICT protocol and topology abstractor to the ICT-agnostic SB management system.

On the primary goals of Chapter 4 was the creation of a virtualization software architecture for real-time emulation of SB. This tool proved that energy management and analytics algorithms in existing building structures could be assessed even without the necessary infrastructure investments. Such software facilitates the financial analysis and energy performance estimation for justifying costly building retrofitting. Hence, the author aims to mitigate the distrust in SB investments with simulation-supported studies on the actual emulated buildings and occupant activities. Unlike the ahead-of-time simulation tools, the emulator executes in real time with building and occupant activities and events and feeds several such inputs in its emulation models. To this day, models of loads, batteries, generators, sensors, actuators, and users have been validated successfully, while others are continuously investigated within the research group. The highlight of this chapter is certainly the immerse parallelism achieved using a cooperative multitasking engine and the emulation theoretical background based on discrete event simulation studies.

Finally, Chapter 5 offers an overview of the entire "Smart Grid Campus Project" and the integration of the SB within. The chapter also assesses how current research on energy management system (EMS) is extending this dissertation's building management system (BMS) for participation in demand side management (DSM) strategies. Therefore, this chapter highlights the great potential of future adaptable SB within the Smart Grid (SG) context.

6.1 Future Work

The SB has a bright future ahead, one that will fundamentally revolutionize the way society considers buildings and urban development in general. The SB will meet the needs of occupants and businesses in a flexible and adaptive manner for sustainable, comfortable, and healthy living spaces. Technological advancements, especially in Internet of Things (IoT), big data, and cybersecurity, coupled with the eventual cost reduction, will catalyze the transformation.

While many aspects of the SB design have been addressed extensively in this dissertation, there exists a couple of elements which merit further investigation and research. Hence, this section proposes several future extensions. They are clustered into two subsections. The short term suggests improvements to the proposed sub-systems architectures, while, the long term one recommends future work as a strategic system-level expansion to the overall SB architecture, in order to further improve its perceived value and address more of the challenges identified in Chapter 1.

6.1.1 Short-term extensions

Smart building modeling and computational system core

To begin with, Chapter 2, which developed the data model of the building, would merit several model extensions. A more detailed model of building generation and storage as well as an extension of the sensor and actuator models are recommended. Furthermore, the chapter demonstrated two modules that are generating high-level data insights: an occupant location tracker and a high-speed thermal simulator for just-in-time heating and energy awareness. These integrated intelligent services should be extended and exposed through the application programming interface (API). Ideal examples of similar intelligent services offer by the SB system are, for example, forecasting of the internal building environment (e.g., temperature, humidity, luminosity, etc.), occupant activity and behavior estimating, load profile forecasting, etc. Finally, an API extension with openADR protocol support is crucial for communication with the energy supplier as required by the EMS.

Distributed message oriented middleware

The proposed distributed middleware architecture is optimized and meets the requirements defined at the beginning of the chapter. Nevertheless, there are still some opportunities for improvement. Firstly, the reliability of the communication can be increased and guaranteed using, when necessary, a delivery acknowledgment scheme. Secondly, the messaging payload formatting, while universal amongst the middleware nodes, is not yet exhaustive. A more standardized version is necessary for defining a comprehensive, holistic, and expandable language for exchanged middleware messages. Finally, while a couple of protocol abstractions have been developed for testing and validation purposes, a practical middleware would require

a bigger "library" of such protocol abstraction modules implemented within the proposed layered software architecture principles.

Building-in-the-loop emulation engine

The building emulator presented in Chapter 4 is a bottom-up, interdisciplinary approach and a holistic solution to SB emulation with real-time physical data input. Multiple software libraries, theoretical models, and design rules were used in the realization of the system. At the time of this writing, the research includes the theoretical background, the design concept, the software architecture and implementation as well as a thorough validation. Nevertheless, there is still room for expansion of such a promising system and the idea of real-time building emulation. The most notable improvement is the research on superior models compatible with this engine. The object-oriented design of the emulation engine and the models' inheritance facilitate the development of more accurate and elaborate models. Therefore, the author has released the source code of the system [239] for initiating follow-up collaboration and expansion within and beyond the research group. Moreover, a quantitative comparison against other building simulation systems, beyond the qualitative one of the state of the art, would significantly highlight the contribution of this work to simulation science in general.

6.1.2 Long-term prospects

It was beyond the scope of this thesis to investigate and develop the management intelligence of the SB. Nevertheless, the author quickly recognized energy providers and occupants as the two prominent actors and stakeholders in the SB's design and operation.

Therefore, while not covered in detail in this dissertation, an advanced EMS is vital for autonomously managing the building's energy while communicating in real time with the energy providers. Such a system is currently in development within the research group and plans to take advantage of the installed BMS in order to optimize the short- and long-term energy consumption and storage while ensuring the comfort of the occupants. In the end, it would be this EMS that would empower integration into the SG and participation in the DSM strategies.

However, managing the occupants' comfort is a nontrivial task. An improved ambient intelligence system, in the form of an occupant preferences and behavior machine learning approach, should be provisioned for extracting the inhabitant-driven parameters and their satisfaction priorities. Those data, combined with the real-time monitored data from the BMS, would be the input to the EMS for augmented decision making and comfort reassurance.

Final thoughts

Finally, as already scrutinized in this dissertation, it is the combination of ambient intelligence, building automation, and energy management that empowers, differentiates, and improves the SB compared to current systems and solutions. The author wishes and hopes that with this systems-thinking dissertation, he has contributed to the literature and the market state of art, not only by means of scientific research, but also with the development and realistic validation of the proposed designs of deployable, scalable, and secure system architectures. Such systems that focus on the mitigation of challenges that hinder SB adoption can ultimately bring the domain's state of the art one step closer to the envisioned and ideal definition of the SB for a sustainable future and urban development.

Bibliography

[1] P. D. United Nations, Department of Economic and Social Affairs, "World Population Prospects: The 2015 Revision, Data Booklet. ST/ESA/SER.A/377," p. 20, 2015.

[2] European Commission, *EU Reference Scenario 2016: Energy, transport and GHG emissions trends to 2050*. European Commission, 2016. [Online]. Available: https://ec.europa.eu/energy/en/data-analysis/energy-modelling

[3] A. Buckman, M. Mayfield, and S. B.M. Beck, "What is a Smart Building?" *Smart and Sustainable Built Environment*, vol. 3, no. 2, pp. 92–109, 2014.

[4] D. Clements-Croome, "Editorial," *Intelligent Buildings International*, vol. 3, no. 4, pp. 221–222, oct 2011.

[5] O. Bozdag, M. Secer, L. Braganca, M. Pinheiro, S. Jalali, R. Mateus, R. Amoeda, and M. C. Guedes, "Energy consumption of RC buildings during their life cycle," *Portugal Sb07 - Sustainable Construction, Materials and Practices: Challenge of the Industry For the New Millennium, Pts 1 and 2*, pp. 480–487, 2007.

[6] D. Clements-Croome, *Intelligent buildings: design, management and operation*. ICE Publishing, 2013.

[7] G. Lilis, G. Conus, N. Asadi, and M. Kayal, "Towards the next generation of intelligent building: An assessment study of current automation and future iot based systems with a proposal for transitional design," *Sustainable Cities and Society*, 2016.

[8] D. H. Meadows, *Thinking in systems: A primer*. Chelsea Green Publishing, 2008.

[9] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication systems for building automation and control," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, 2005.

[10] A. Ghaffarianhoseini, U. Berardi, H. AlWaer, S. Chang, E. Halawa, A. Ghaffarianhoseini, and D. Clements-Croome, "What is an intelligent building? analysis of recent interpretations from an international perspective," *Architectural Science Review*, vol. 59, no. 5, pp. 338–357, 2016.

[11] R. I. Ogie, P. Perez, and V. Dignum, "Smart infrastructure: an emerging frontier for multidisciplinary research," *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, vol. 170, no. SC1, pp. 8–16, 2017.

[12] Z. J. Yu Zhun Jerry, F. Haghighat, B. C. M. Fung, E. Morofsky, and H. Yoshino, "A methodology for identifying and improving occupant behavior in residential buildings," *Energy*, vol. 36, no. 11, pp. 6596–6608, 2011.

[13] N. N. Kang, S. H. Cho, and J. T. Kim, "The energy-saving effects of apartment residents' awareness and behavior," *Energy and Buildings*, vol. 46, pp. 112–122, 2012.

[14] P. W. Schultz, M. Estrada, J. Schmitt, R. Sokoloski, and N. Silva-Send, "Using in-home displays to provide smart meter feedback about household electricity consumption: A randomized control trial comparing kilowatts, cost, and social norms," *Energy*, vol. 90, pp. 351–358, 2015.

[15] G. Peschiera, J. E. Taylor, and J. A. Siegel, "Response-relapse patterns of building occupant electricity consumption following exposure to personal, contextualized and occupant peer network utilization data," *Energy and Buildings*, vol. 42, no. 8, pp. 1329–1336, 2010.

[16] J. Mankoff, S. R. Fussell, T. Dillahunt, R. Glaves, C. Grevet, M. Johnson, D. Matthews, H. S. Matthews, R. Mcguire, R. Thompson, A. Shick, and L. Setlock, "StepGreen.org: Increasing Energy Saving Behaviors via Social Networks," *Proceedings of the International AAAI Conference on Weblogs and Social Media (ICWSM 2008)*, no. 2008, pp. 106–113, 2010.

[17] R. K. Jain, R. Gulbinas, J. E. Taylor, and P. J. Culligan, "Can social influence drive energy savings? Detecting the impact of social influence on the energy consumption behavior of networked users exposed to normative eco-feedback," *Energy and Buildings*, vol. 66, pp. 119–127, 2013.

[18] D. Clements-Croome, "Sustainable intelligent buildings for people: A review," *Intelligent Buildings International*, vol. 3, no. 2, p. 67, 2011.

[19] H. Alwaer and D. Clements-Croome, "Key performance indicators (kpis) and priority setting in using the multi-attribute approach for assessing sustainable intelligent buildings," *Building and Environment*, vol. 45, no. 4, pp. 799–807, 2010.

[20] N. Balta-Ozkan, R. Davidson, M. Bicket, and L. Whitmarsh, "Social barriers to the adoption of smart homes," *Energy Policy*, vol. 63, pp. 363–374, 2013.

[21] W. Granzer, W. Kastner, and C. Reinisch, "Gateway-free integration of BACnet and KNX using multi-protocol devices," *IEEE International Conference on Industrial Informatics (INDIN)*, pp. 973–978, 2008.

[22] BACnet International, "BACnet Certification," 2015. [Online]. Available: http://www.webcitation.org/6b69SRwwN

[23] KNX Association, "KNX ETS4 licence," 2015. [Online]. Available: http://www.webcitation.org/6b68x0s3f

[24] A. McMillan. (2010) The Cost(s) of BACnet. [Online]. Available: http://www.webcitation.org/6axEpwc30

[25] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.

[26] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[27] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle, "Security challenges in the IP-based Internet of Things," *Wireless Personal Communications*, vol. 61, pp. 527–542, 2011.

[28] X. Teng, J. B. Wendt, and M. Potkonjak, "Security of IoT systems: Design challenges and opportunities," in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, 2014, pp. 417–423.

[29] R. H. Weber, "Internet of Things - New security and privacy challenges," *Computer Law and Security Review*, vol. 26, no. 1, pp. 23–30, 2010.

[30] J. Cao, B. Carminati, E. Ferrari, and K. L. Tan, "CASTLE: Continuously anonymizing data streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 337–352, 2011.

[31] J. K. W. Wong, H. Li, and S. W. Wang, "Intelligent building research: A review," *Automation in Construction*, vol. 14, pp. 143–159, 2005.

[32] Gartner, "Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business," 2014. [Online]. Available: http://www.webcitation.org/6b5yJycdt

[33] ——, "Gartner's 2015 Hype Cycle for Emerging Technologies Identifies the Computing Innovations That Organizations Should Monitor," 2015. [Online]. Available: http://www.webcitation.org/6b5yriXxV

[34] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home automation in the wild," *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, p. 2115, 2011.

[35] L. Li, Z. Jin, G. Li, L. Zheng, and Q. Wei, "Modeling and analyzing the reliability and cost of service composition in the IoT: A probabilistic approach," *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, pp. 584–591, 2012.

[36] H. Madsen, G. Albeanu, B. Burtschy, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," *International Conference on Systems, Signals, and Image Processing*, pp. 43–46, 2013.

[37] A.-G. Paetz, E. Dütschke, and W. Fichtner, "Smart Homes as a Means to Sustainable Energy Consumption: A Study of Consumer Perceptions," *Journal of Consumer Policy*, vol. 35, no. 1, pp. 23–41, 2012.

[38] P. Palensky and D. Dietrich, "Demand Side Management: Demand Response, Intelligent Energy Systems, and Smart Loads," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 3, pp. 381–388, 2011.

[39] S. Nolan and M. O'Malley, "Challenges and barriers to demand response deployment and evaluation," *Applied Energy*, vol. 152, pp. 1–10, 2015.

[40] N. Oconnell, P. Pinson, H. Madsen, and M. Omalley, "Benefits and challenges of electrical demand response: A critical review," *Renewable and Sustainable Energy Reviews*, vol. 39, pp. 686–699, 2014.

[41] B. T. Samad, E. Koch, and P. Stluka, "Automated Demand Response for Smart Buildings and Microgrids: The State of the Practice and Research Challenges," *Proceedings of the IEEE*, vol. 104, no. 4, pp. 726–744, 2016.

[42] J. Han and M. a. Piette, "Solutions for summer electric power shortages : Demand Response and its applications in air conditioning and refrigerating systems," *Refrigeration, Air Conditioning, & Electric Power Machinery*, vol. 29, no. 1, pp. 1–4, 2008.

[43] C. Fischer, "Feedback on household electricity consumption: A tool for saving energy?" *Energy Efficiency*, vol. 1, no. 1, pp. 79–104, 2008.

[44] R. K. Jain, J. E. Taylor, and P. J. Culligan, "Investigating the impact eco-feedback information representation has on building occupant energy consumption behavior and savings," *Energy and Buildings*, vol. 64, pp. 408–414, 2013.

[45] K. Buchanan, R. Russo, and B. Anderson, "The question of energy reduction: The problem(s) with feedback," *Energy Policy*, vol. 77, pp. 89–96, 2015.

[46] A. Nilsson, C. J. Bergstad, L. Thuvander, D. Andersson, K. Andersson, and P. Meiling, "Effects of continuous feedback on households' electricity consumption: Potentials and barriers," *Applied Energy*, vol. 122, pp. 17–23, 2014.

[47] L. Pereira, F. Quintal, M. Barreto, and N. J. Nunes, "Understanding the limitations of eco-feedback: A one-year long-term study," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7947 LNCS, pp. 237–255, 2013.

[48] Z. Yu, B. C. Fung, F. Haghighat, H. Yoshino, and E. Morofsky, "A systematic procedure to study the influence of occupant behavior on building energy consumption," *Energy and Buildings*, vol. 43, no. 6, pp. 1409–1417, jun 2011.

[49] T. Hong, S. C. Taylor-Lange, S. D'Oca, D. Yan, and S. P. Corgnati, "Advances in research and applications of energy-related occupant behavior in buildings," *Energy and Buildings*, vol. 116, pp. 694–702, 2016.

[50] T. A. Nguyen and M. Aiello, "Energy intelligent buildings based on user activity: A survey," *Energy and Buildings*, vol. 56, pp. 244–257, 2013.

[51] M. Gupta, S. S. Intille, and K. Larson, "Adding GPS-control to traditional thermostats: An exploration of potential energy savings and design challenges," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5538 LNCS, pp. 95–114, 2009.

[52] W. Kleiminger, F. Mattern, and S. Santini, "Predicting household occupancy for smart heating control: A comparative performance analysis of state-of-the-art approaches," *Energy and Buildings*, vol. 85, pp. 493–505, 2014.

[53] L. Klein, J. Y. Kwak, G. Kavulya, F. Jazizadeh, B. Becerik-Gerber, P. Varakantham, and M. Tambe, "Coordinating occupant behavior for building energy and comfort management using multi-agent systems," *Automation in Construction*, vol. 22, pp. 525–536, mar 2012.

[54] R. Missaoui, H. Joumaa, S. Ploix, and S. Bacha, "Managing energy Smart Homes according to energy prices: Analysis of a Building Energy Management System," *Energy and Buildings*, vol. 71, pp. 155–167, 2014.

[55] S. T. Bushby, "BACnet™: a standard communication infrastructure for intelligent buildings," *Automation in Construction*, vol. 6, no. 5-6, pp. 529–540, 1997.

[56] KNX Association, "KNX Specifications version 2.1," 2014. [Online]. Available: http://www.knx.org/

[57] S. Wang, Z. Xu, H. Li, J. Hong, and W. Z. Shi, "Investigation on intelligent building standard communication protocols and application of IT technologies," *Automation in Construction*, vol. 13, no. 5, pp. 607–619, sep 2004.

[58] S. Wang and J. Xie, "Integrating Building Management System and facilities management on the Internet," *Automation in Construction*, vol. 11, no. 6, pp. 707–715, oct 2002.

[59] D. Loy, D. Dietrich, and S. Hans-Joerg, *Open Control Networks: LonWorks/EIA 709 Technology*. Kluwer Academic Publishers, 2001.

[60] H. Jarvinen, A. Litvinov, and P. Vuorimaa, "Integration platform for home and building automation systems," *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, no. PerNets, pp. 292–296, 2011.

[61] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, "Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture," *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pp. 1361–1367, 2013.

[62] W. Kastner, M. Kofler, M. Jung, G. Gridling, and J. Weidinger, "Building automation systems integration into the Internet of Things the IoT6 approach, its realization and validation," *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014*, 2014.

[63] C. Reinisch, W. Granzer, F. Praus, and W. Kastner, "Integration of heterogeneous building automation systems using ontologies," in *2008 34th Annual Conference of IEEE Industrial Electronics*. IEEE, nov 2008, pp. 2736–2741.

[64] J. Han, Y. Jeong, and I. Lee, "Efficient building energy management system based on ontology, inference rules, and simulation," in *International Conference on Intelligent Building and Management*, 2011.

[65] J. Ploennigs, B. Hensel, H. Dibowski, and K. Kabitzsch, "BASont - A modular, adaptive building automation system ontology," *IECON Proceedings (Industrial Electronics Conference)*, pp. 4827–4833, 2012.

[66] G. Fortino, A. Guerrieri, G. M. P. Ohare, and A. Ruzzelli, "A flexible building management framework based on wireless sensor and actuator networks," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1934–1952, 2012.

[67] C. Farias, H. Soares, L. Pirmez, F. Delicato, I. Santos, L. F. Carmo, J. Souza, A. Zomaya, and M. Dohler, "A control and decision system for smart buildings using wireless sensor and actuator networks," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 120–135, 2014.

[68] J. R. Gisbert, C. Palau, M. Uriarte, G. Prieto, J. A. Palazón, M. Esteve, O. López, J. Correas, M. C. Lucas-Estañ, P. Giménez, A. Moyano, L. Collantes, J. Gozálvez, B. Molina, O. Lázaro, and A. González, "Integrated system for control and monitoring industrial wireless networks for labor risk prevention," *Journal of Network and Computer Applications*, vol. 39, no. 1, pp. 233–252, 2014.

[69] T. G. Stavropoulos, A. Tsioliaridou, G. Koutitas, D. Vrakas, and I. Vlahavas, "System architecture for a smart university building," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6354 LNCS, pp. 477–482, 2010.

[70] A. De Paola, S. Gaglio, G. Lo Re, and M. Ortolani, "Sensor 9k: A testbed for designing and experimenting with WSN-based ambient intelligence applications," *Pervasive and Mobile Computing*, vol. 8, no. 3, pp. 448–466, 2012.

[71] E. Z. Tragos, M. Foti, M. Surligas, G. Lambropoulos, S. Pournaras, S. Papadakis, and V. Angelakis, "An IoT based intelligent building management system for ambient assisted living," *2015 IEEE International Conference on Communication Workshop (ICCW)*, pp. 246–252, 2015.

[72] L. W. Yeh, Y. C. Wang, and Y. C. Tseng, "iPower: an energy conservation system for intelligent buildings by wireless sensor networks," *International Journal of Sensor Networks*, vol. 5, p. 1, 2009.

[73] M. Weiss and D. Guinard, "Increasing energy awareness through web-enabled power outlets," *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia - MUM '10*, pp. 1–10, 2010.

[74] I. Hong, J. Byun, and S. Park, "Cloud computing-based building energy management system with ZigBee sensor network," *Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012*, pp. 547–551, 2012.

[75] P. Zhao, M. G. Simoes, and S. Suryanarayanan, "A conceptual scheme for cyber-physical systems based energy management in building structures," *2010 9th IEEE/IAS International Conference on Industry Applications - INDUSCON 2010*, pp. 1–6, 2010.

[76] T. Gamauf, T. Leber, K. Pollhammer, and F. Kupzog, "A generalized load management gateway coupling smart buildings to the grid," in *IEEE AFRICON Conference*, 2011, pp. 1–5.

[77] G. Anastasi, F. Corucci, and F. Marcelloni, "An intelligent system for electrical energy management in buildings," *International Conference on Intelligent Systems Design and Applications, ISDA*, pp. 702–707, nov 2011.

[78] H.-Y. Huang, J.-Y. Yen, S.-L. Chen, and F.-C. Ou, "Development of an Intelligent Energy Management Network for Building Automation," *IEEE Transactions on Automation Science and Engineering*, vol. 1, no. 1, pp. 14–25, 2004.

[79] H. Chen, P. Chou, S. Duri, H. Lei, and J. Reason, "The design and implementation of a smart building control system," *Proceedings - IEEE International Conference on e-Business Engineering, ICEBE 2009; IEEE Int. Workshops - AiR 2009; SOAIC 2009; SOKMBI 2009; ASOC 2009*, pp. 255–262, 2009.

[80] A. Capone, M. Barros, H. Hrasnica, and S. Tompros, "A new architecture for reduction of energy consumption of home appliances," in *TOWARDS eENVIRONMENT, European conference of the Czech Presidency of the Council of the EU*, 2009, pp. 1–8.

[81] "openHAB," 2017. [Online]. Available: https://www.openhab.org/

[82] "FHEM: home automation server." [Online]. Available: http://fhem.de/fhem.html

[83] "Domoticz: home automation." [Online]. Available: https://domoticz.com/

[84] Eclipse IoT, "Eclipse Smarthome: A Flexible Framework for the Smart Home," 2015. [Online]. Available: https://www.eclipse.org/smarthome/

[85] "Home Assistant: open source home automation." [Online]. Available: https://home-assistant.io/

[86] Pacific Northwest National Laboratory, "Volttron Platform," 2016. [Online]. Available: http://transactionalnetwork.pnnl.gov/volttron.stm

[87] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[88] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services," *Proceeding of the 17th international conference on World Wide Web - WWW '08*, p. 805, 2008.

[89] InfluxData, "InfluxDB - Open-Source Time Series Database," 2017. [Online]. Available: https://github.com/influxdata/influxdb

[90] "OpenTSDB - A Distributed, Scalable Monitoring System," 2017. [Online]. Available: http://opentsdb.net/

[91] T. Oetiker, "RRDtool," 2016. [Online]. Available: http://oss.oetiker.ch/rrdtool/

[92] F. Mattern, T. Staake, and M. Weiss, "ICT for green – How Computers Can Help Us to Conserve Energy," *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking - e-Energy '10*, p. 1, 2010.

[93] A. Spagnolli, N. Corradi, L. Gamberini, E. Hoggan, G. Jacucci, C. Katzeff, L. Broms, and L. Jonsson, "Eco-feedback on the go: Motivating energy awareness," *Computer*, vol. 44, no. 5, pp. 38–45, 2011.

[94] M. Zeifman, "Disaggregation of home energy display data using probabilistic approach," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 23–31, 2012.

[95] C. Chen and D. J. Cook, "Behavior-based home energy prediction," *Proceedings - 8th International Conference on Intelligent Environments, IE 2012*, pp. 57–63, 2012.

[96] B. J. Birt, G. R. Newsham, I. Beausoleil-Morrison, M. M. Armstrong, N. Saldanha, and I. H. Rowlands, "Disaggregating categories of electrical energy end-use from whole-house hourly data," *Energy and Buildings*, vol. 50, pp. 93–102, jul 2012.

[97] M. Maasoumy, A. Pinto, and A. Sangiovanni-Vincentelli, "Model-Based Hierarchical Optimal Control Design for HVAC Systems," *ASME 2011 Dynamic Systems and Control Conference and Bath/ASME Symposium on Fluid Power and Motion Control, Volume 1*, pp. 271–278, 2011.

[98] G. Fraisse, C. Viardot, O. Lafabrie, and G. Achard, "Development of a simplified and accurate building model based on electrical analogy," *Energy and Buildings*, vol. 34, pp. 1017–1031, 2002.

[99] D. Gyalistras and M. Gwerder, "Use of weather and occupancy forecasts for optimal building climate control (OptiControl): Two years progress report," ETH, Zurich, Tech. Rep. September, 2009.

[100] B. Lehmann, D. Gyalistras, M. Gwerder, K. Wirth, and S. Carl, "Intermediate complexity model for Model Predictive Control of Integrated Room Automation," *Energy and Buildings*, vol. 58, pp. 250–262, 2013.

[101] D. Sturzenegger, D. Gyalistras, V. Semeraro, M. Morari, and R. S. Smith, "BRCM Matlab Toolbox : Model Generation for Model Predictive Building Control," *American Control Conference*, 2014.

[102] Linear Technology, "LTspice," 2015. [Online]. Available: http://www.linear.com/designtools/software/

[103] M. Gwerder and D. Gyalistras, "Final Report : Use of Weather And Occupancy Forecasts For Optimal Building Climate Control – Part II : Demonstration (OptiControl-II)," ETH Zürich, Tech. Rep. September, 2013.

[104] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 37, no. 6, pp. 1067–1080, 2007.

[105] R. Mautz, "Indoor Positioning Technologies," p. 127, 2012.

[106] Qualcomm, "gpsOne Hybrid Position Location Technology."

[107] Skyhook Wireless, "Skyhook," 2015. [Online]. Available: http://www.skyhookwireless.com/

[108] S. Zirari, P. Canalda, and F. Spies, "WiFi GPS based combined positioning algorithm," *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, no. Ea 4269, pp. 684–688, 2010.

[109] M. Kjærgaard, H. Blunck, T. Godsk, T. Toftkjær, D. Christensen, and K. Grønbæk, "Indoor positioning using GPS revisited," *Pervasive Computing*, vol. 45, no. 1, pp. 38–56, 2010.

[110] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Communications Surveys and Tutorials*, vol. 11, no. 1, pp. 13–32, 2009.

[111] S. S. Saad and Z. S. Nakad, "A standalone RFID indoor positioning system using passive tags," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 5, pp. 1961–1970, 2011.

[112] L. Ni and A. Patil, "LANDMARC: indoor location sensing using active RFID," *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003).*, pp. 407–415, 2003.

[113] Google Inc., "Fused Location Provider Api," 2015. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi

[114] ——, "Geofencing Api," 2015. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/GeofencingApi

[115] C. Koehler, B. Ziebart, J. Mankoff, and A. Dey, "TherML: occupancy prediction for thermostat control," *Proceedings of the 2013 . . .*, pp. 103–112, 2013.

[116] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng, "From wireless sensor networks towards cyber physical systems," *Pervasive and Mobile Computing*, vol. 7, no. 4, pp. 397–413, 2011.

[117] J. Kleissl and Y. Agarwal, "Cyber-physical energy systems: Focus on smart buildings," *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 749–754, 2010.

[118] A. Savvides, I. Paschalidis, and M. Caramanis, "Cyber-physical systems for next generation intelligent buildings," *ACM SIGBED Review*, vol. 8, no. 2, pp. 35–38, 2011.

[119] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Dordrecht: Springer Netherlands, 2011.

[120] J. Wong and H. Li, "Development of a conceptual model for the selection of intelligent building systems," *Building and Environment*, vol. 41, no. 8, pp. 1106–1123, 2006.

[121] U. Wilke, F. Haldi, J.-L. Scartezzini, and D. Robinson, "A bottom-up stochastic model to predict building occupants' time-dependent activities," *Building and Environment*, vol. 60, pp. 254–264, feb 2013.

[122] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming sensor networks into multi-application sensing infrastructures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7158 LNCS, pp. 65–81, 2012.

[123] E. A. Lee, "Cyber Physical Systems: Design Challenges," *Proc. of 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'08)*, pp. 363–369, 2008.

[124] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer US, 2011.

[125] S. Hadim and N. Mohamed, "Middleware: Middleware challenges and approaches for wireless sensor networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 1–23, 2006.

[126] M. Wang, J. Cao, J. Li, and S. Dasi, "Middleware for wireless sensor networks: A survey," *Journal of Computer Science and Technology*, vol. 23, no. 2006, pp. 305–326, 2008.

[127] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling technologies," *IEEE Internet of Things Journal*, vol. X, no. X, pp. 1–1, 2016.

[128] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[129] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the internet of things," in *Proceedings of the 2012 International Conference on Collaboration Technologies and Systems, CTS 2012*, 2012, pp. 21–26.

[130] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18, no. 1, pp. 15–21, 2004.

[131] M. M. Molla and S. I. Ahamed, "A survey of middleware for sensor network and challenges," in *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, 2006, pp. 6 pp.–228.

[132] E. P. D. Freitas, "A survey on adaptable middleware for wireless sensor networks," *Halmstad University*, no. August, pp. 9–52, 2008.

[133] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," *Service Oriented Computing and Applications*, vol. 5, no. 2, pp. 71–85, 2011.

[134] P. Pietzuch and J. Bacon, "Hermes: a distributed event-based middleware architecture," in *Proceedings of 22nd International Conference on Distributed Computing Systems Workshops*, 2002.

[135] P. R. Pietzuch, "Hermes: A scalable event-based middleware," Ph.D. dissertation, University of Cambridge, 2004.

[136] Y. Liu, I. Gorton, L. Bass, C. Hoang, and S. Abanmi, "MEMS: A method for evaluating middleware architectures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4214 LNCS, p. 9, 2006.

[137] M. Martonosi and T. Liu, "Impala : A Middleware System for Managing Autonomic , Parallel Sensor Systems," *ACM SIGPLAN Notices*, vol. 38, no. 10, pp. 107–118, 2003.

[138] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.

[139] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and U. C. Berkeley, "TinyDB: An Acquisitional Query Processing System for Sensor Networks 1," *Database*, vol. V, no. 212, 2004.

[140] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record*, vol. 31, no. 3, pp. 9–18, 2002.

[141] J. R. Votano, M. Parham, and L. H. Hall, "Sensor Information Networking Architecture and Applications," *IEEE Personal Communications*, vol. 1, no. August 2001, pp. 52–59, 2001.

[142] "LinkSmart Middleware." [Online]. Available: https://www.linksmart.eu/redmine

[143] M. Eisenhauer, P. Rosengren, and P. Antolin, "A development platform for integrating wireless devices and sensors into Ambient Intelligence systems," in *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, SECON Workshops 2009*, vol. 00, no. c, 2009, pp. 1–3.

[144] A. Boulis, C. C. Han, R. Shea, and M. B. Srivastava, "SensorWare: Programming sensor networks beyond code update and querying," *Pervasive and Mobile Computing*, vol. 3, no. 4, pp. 386–412, 2007.

[145] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: A publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37–44, 2006.

[146] P. Evensen and H. Meling, "Sensewrap: A service oriented middleware with sensor virtualization and self-configuration," in *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, Dec. 2009, pp. 261–266.

[147] E. Avilés-López and J. A. García-Macías, "TinySOA: A service-oriented architecture for wireless sensor networks," *Service Oriented Computing and Applications*, vol. 3, no. 2, pp. 99–108, 2009.

[148] C. L. Fok, G. C. Roman, and C. Lu, "Servilla: A flexible service provisioning middleware for heterogeneous sensor networks," *Science of Computer Programming*, vol. 77, no. 6, pp. 663–684, 2012.

[149] S. Wang, Z. Xu, J. Cao, and J. Zhang, "A middleware for web service-enabled integration and interoperation of intelligent building systems," *Automation in Construction*, vol. 16, no. 1, pp. 112–121, jan 2007.

[150] P. Arjunan, N. Batra, H. Choi, and A. Singh, "SensorAct : A Privacy and Security Aware Federated Middleware for Building Management," *BuildSys '12*, pp. 80–87, 2012.

[151] L. F. Ducreux, C. Guyon-Gardeux, S. Lesecq, F. Pacull, and S. R. Thior, "Resource-based middleware in the context of heterogeneous building automation systems," in *IECON Proceedings (Industrial Electronics Conference)*, 2012, pp. 4847–4852.

[152] T. Le Guilly, P. Olsen, A. P. Ravn, J. B. Rosenkilde, and A. Skou, "HomePort: Middleware for heterogeneous home automation networks," in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2013*, no. March, 2013, pp. 627–633.

[153] E. Patti, A. Acquaviva, M. Jahn, F. Pramudianto, R. Tomasi, D. Rabourdin, J. Virgone, and E. Macii, "Event-driven user-centric middleware for energy-efficient buildings and public spaces," *Systems Journal IEEE*, vol. 99, no. 3, pp. 1–10, 2014.

[154] G. Banavar, T. Chandra, R. Strom, and D. Sturman, "A Case for Message Oriented Middleware," *Distributed Computing – 13th International Symposium, DISC'99*, pp. 1–17, 1999.

[155] W. Emmerich, "Software Engineering and Middleware : A Roadmap," *Communications of the ACM,*, p. 11, 2000.

[156] OMG, "CORBA: Common Object Request Broker Architecture," 2013. [Online]. Available: http://www.corba.org/

[157] M. Henning, "A new approach to object-oriented middleware," *IEEE Internet Computing*, vol. 8, no. 1, pp. 66–75, 2004.

[158] A. Wollrath, R. Riggs, and J. Waldo, "A Distributed Object Model for the Java System," in *COOTS'96 Proceedings of the 2nd conference on USENIX Conference on Object-Oriented Technologies*, vol. 9, no. 4, 1996.

[159] A. T. Campbell, G. Coulson, and M. E. Kounavis, "Managing Complexity: Middleware Explained." *IT Professional*, vol. 1, p. 22, 1999.

[160] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[161] J. Al-Jaroodi and N. Mohamed, "Service-oriented middleware: A survey," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 211–220, 2012.

[162] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook White Paper*, p. 8, 2007.

[163] A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, and M. Sobczak, "Middleware Trends And Market Leaders 2011," in *13th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2011)*, vol. C111010, 2011, p. 4.

[164] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.

[165] P. T. H. Eugster, P. a. Felber, A.-m. Kermarrec, R. Guerraoui, and A.-m. Kermarrec, "The Many Faces of Publish / Subscribe," *Computing*, vol. 35, no. 2, pp. 114–131, 2003.

[166] P. Hintjens, "What is wrong with AMQP," 2008. [Online]. Available: http://www.imatix.com/articles:whats-wrong-with-amqp/

[167] ——, *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, Inc., 2013.

[168] G. Sharma, S. Bala, and A. K. Verma, "Security Frameworks for Wireless Sensor Networks-Review," *Procedia Technology*, vol. 6, pp. 978–987, 2012.

[169] J. Al-Jaroodi, I. Jawhar, A. Al-Dhaheri, F. Al-Abdouli, and N. Mohamed, "Security middleware approaches and issues for ubiquitous applications," *Computers and Mathematics with Applications*, vol. 60, no. 2, pp. 187–197, 2010.

[170] Z. Liu and D. Peng, "A security-supportive middleware architecture for pervasive computing," *Proceedings - 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing, DASC 2006*, pp. 137–144, 2006.

[171] Z. Jiang, K. Lee, S. Kim, H. Bae, S. Kim, and S. Kang, "Design of a security management middleware in ubiquitous computing environments," *Parallel and Distributed Computing, Applications and Technologies, PDCAT Proceedings*, vol. 2005, pp. 306–308, 2005.

[172] A. Marín, W. Mueller, R. Schaefer, F. Almenarez, D. Díaz, and M. Ziegler, "Middleware for secure home access and control," in *Proceedings - Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2007*, 2007, pp. 489–494.

[173] S. I. Ahamed, M. M. Haque, and K. M. I. Asif, "S-MARKS: A middleware secure by design for the pervasive computing environment," in *Proceedings - International Conference on Information Technology-New Generations, ITNG 2007*, 2007, pp. 303–308.

[174] M. Zhang, S. Zhu, B. Yang, and W. Zhang, "Trust-based distributed authentication middleware in ubiquitous mobile environments," in *Proceedings - Third International Conference on Natural Computation, ICNC 2007*, vol. 5, no. Icnc, 2007, pp. 814–818.

[175] S. Khanvilkar and A. Khokhar, "Virtual private networks: An overview with performance evaluation," *IEEE Communications Magazine*, vol. 42, no. 10, pp. 146–154, 2004.

[176] "Unencapsulated MS-CHAP v2 Authentication Could Allow Information Disclosure," 2012. [Online]. Available: https://technet.microsoft.com/en-us/library/security/2743314.aspx

[177] B. Schneier and Mudge, "Cryptanalysis of Microsoft's point-to-point tunneling protocol (PPTP)," in *5th ACM Conference on Computer and Communications Security*, 1998, pp. 132–141.

[178] S. Narayan, S. S. Kolahi, K. Brooking, and S. de Vere, "Performance Evaluation of Virtual Private Network Protocols in Windows 2003 Environment," in *2008 International Conference on Advanced Computer Theory and Engineering*, 2008, pp. 69–73.

[179] A. A. Joha, F. B. Shatwan, and M. Ashibani, "Performance evaluation for remote access VPN on Windows server 2003 and fedora core 6," in *8th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, TELSIKS 2007, Proceedings of Papers*, 2007, pp. 587–592.

[180] A. Hoban, "Using Intel ® AES New Instructions and PCLMULQDQ to Significantly Improve IPSec Performance on Linux," *Intel Corporation*, no. August, pp. 1–26, 2010.

[181] I. Kotuliak, P. Rybár, and P. Trúchly, "Performance comparison of IPsec and TLS based VPN technologies," in *ICETA 2011 - 9th IEEE International Conference on Emerging eLearning Technologies and Applications, Proceedings*, 2011, pp. 217–221.

[182] J. B. R. Lawas, A. C. Vivero, and A. Sharma, "Network performance evaluation of VPN protocols (SSTP and IKEv2)," *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, pp. 1–5, 2016.

[183] T. Nie and T. Zhang, "A study of DES and Blowfish encryption algorithm," in *TENCON 2009*. IEEE, nov 2009, pp. 1–4.

[184] A. Ramesh and A. Suruliandi, "Performance analysis of encryption algorithms for Information Security," in *2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT)*. Institute of Electrical and Electronics Engineers (IEEE), mar 2013, pp. 840–844.

[185] G. Lilis, G. Conus, and M. Kayal, "A distributed, event-driven building management platform on web technologies," in *1st International Conference on Event-Based Control, Communication, and Signal Processing*, 2015.

[186] L. Gelazanskas and K. A. A. Gamage, "Demand side management in smart grid: A review and proposals for future direction," *Sustainable Cities and Society*, vol. 11, pp. 22–30, 2014.

[187] C. W. Gellings and J. H. Chamberlin, *Demand-side management : concepts and methods*. Fairmont Press, 1993.

[188] P. Palensky, E. Widl, and A. Elsheikh, "Simulating cyber-physical energy systems: Challenges, tools and methods," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 3, pp. 318–326, 2014.

[189] C. Talcott, "Cyber-physical systems and events," in *Software-Intensive Systems and New Computing Paradigms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 101–115.

[190] A. T. Al-Hammouri, "A comprehensive co-simulation platform for cyber-physical systems," *Computer Communications*, vol. 36, no. 1, pp. 8–19, Dec. 2012.

[191] A. Banerjee, J. Banerjee, G. Varsamopoulos, Z. Abbasi, and S. K. S. Gupta, "Hybrid simulator for cyber-physical energy systems," *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013*, pp. 1–6, May 2013.

[192] B. Wang and J. S. Baras, "Hybridsim: A modeling and co-simulation toolchain for cyber-physical systems," *Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pp. 33–40, 2013.

[193] P. Riederer, "Matlab/simulink for building and hvac simulation - state of the art," *IBPSA 2005 - International Building Performance Simulation Association 2005*, pp. 1019–1026, 2005.

[194] J. Venkatesh, B. Aksanli, J. C. Junqua, P. Morin, and T. S. Rosing, "Homesim: Comprehensive, smart, residential electrical energy simulation and scheduling," in *2013 International Green Computing Conference Proceedings, IGCC 2013*. IEEE, Jun. 2013, pp. 1–8.

[195] J. Park, M. Moon, S. Hwang, and K. Yeom, "Cass: A context-aware simulation system for smart home," in *5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)*. IEEE, Aug. 2007, pp. 461–467.

[196] M. Drăgoicea, L. Bucur, and M. Pătraşcu, "A service oriented simulation architecture for intelligent building management," in *Exploring Services Science*, 2013, pp. 14–28.

[197] N. Martínez, J.-F. Martínez, and V. Hernández Díaz, "Virtualization of event sources in wireless sensor networks for the internet of things," *Sensors*, vol. 14, no. 12, Dec. 2014.

[198] A. Merentitis and F. Zeiger, "Wsn trends: Sensor infrastructure virtualization as a driver towards the evolution of the internet of things," in *UBICOMM 2013 : The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, no. c, 2013, pp. 113–118.

[199] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, and T. Abdelzaher, "Virtual battery: An energy reserve abstraction for embedded sensor networks," in *2008 Real-Time Systems Symposium*. IEEE, Nov. 2008, pp. 123–133.

[200] J. He, Y. Geng, Y. Wan, S. Li, and K. Pahlavan, "A cyber physical test-bed for virtualization of rf access environment for body sensor network," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3826–3836, Oct. 2013.

[201] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 553–576, Jan. 2016.

[202] B. De Carolis, G. Cozzolongo, S. Pizzutilo, and V. L. Plantamura, "Agent-based home simulation and control," in *Foundations of Intelligent Systems*, no. April 2016. Springer Nature, 2005, pp. 404–412.

[203] G. Morganti, A. M. Perdon, G. Conte, and D. Scaradozzi, "Multi-agent system theory for modelling a home automation system," in *10^{th} International Work-Conference on Artificial Neural Networks*, 2009, pp. 585–593.

[204] P. Zeigler, H. Praehofer, and T. Kim, *Theory of Modeling and Simulation*, 2nd ed. New York: Academic Press, 2000.

[205] J. S. Hong, H.-S. Song, T. G. Kim, and K. H. Park, "A real-time discrete event system specification formalism for seamless real-time software development," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 7, no. 4, pp. 355–375, 1997.

[206] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*. ICST, 2008, pp. 60:1—-60:10.

[207] F. Howell and R. Mcnab, "simjava: A discrete event simulation library for java," *Computer*, vol. 30, pp. 51–56, 1998.

[208] N. Matloff, "Introduction to discrete-event simulation and the simpy language," *Davis, CA. Dept of Computer Science. University*, pp. 1–33, 2008.

[209] K. Helsgaun, "Discrete event simulation in java," Department of Computer Science Roskilde University, Denmark, Tech. Rep., 2000.

[210] K. S. Perumalla and R. M. Fujimoto, "Efficient large-scale process-oriented parallel simulations," in *WSC '98 Proceedings of the 30^{th} conference on Winter simulation*, 1998, pp. 459–466.

[211] P. O. Siebers, C. M. Macal, J. Garnett, D. Buxton, and M. Pidd, "Discrete-event simulation is dead, long live agent-based simulation!" *Journal of Simulation*, vol. 4, no. 3, pp. 204–210, 2010.

[212] A. Gustafsson, "Threads without the pain," *Queue*, vol. 3, no. 9, p. 34, 2005.

[213] Q. Fan and Q. Wang, "Performance comparison of web servers with different architectures: A case study using high concurrency workload," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE, Nov. 2015, pp. 37–42.

[214] D. Pariag, T. Brecht, A. Harji, P. Buhr, A. Shukla, and D. R. Cheriton, "Comparing the performance of web server architectures," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, p. 231, Jun. 2007.

[215] F. Feng, R. Lu, and C. Zhu, "A combined state of charge estimation method for lithium-ion batteries used in a wide ambient temperature range," *Energies*, vol. 7, no. 5, pp. 3004–3032, 2014.

[216] R. P. Sera D Teodorescu R, "Pv panel model based on datasheets values," *IEEE Transactions on Power Electronics*, no. 4, pp. 2392–2396, 2007.

[217] I. M. Syed and A. Yazdani, "Simple mathematical model of photovoltaic module for simulation in matlab/simulink," in *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, no. I. IEEE, May 2014, pp. 1–6.

[218] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *Proceedings - IEEE Military Communications Conference MILCOM*. IEEE, Nov. 2008, pp. 1–7.

[219] K. Fall and K. Varadhan, "The network simulator (ns-2)," 2007. [Online]. Available: http://www.isi.edu/nsnam/ns/

[220] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, 2006.

[221] X. Chang, "Network simulations with opnet," *Simulation Conference Proceedings, 1999 Winter*, vol. 1, pp. 307–314 vol.1, 1999.

[222] I. S. Hammoodi, B. G. Stewart, A. Kocian, and S. G. McMeekin, "A comprehensive performance study of opnet modeler for zigbee wireless sensor networks," *NGMAST 2009 - 3rd International Conference on Next Generation Mobile Applications, Services and Technologies*, pp. 357–362, 2009.

[223] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: A library for parallel simulation of large-scale wireless networks," in *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation*. Banff, Alberta, Canada: IEEE Computer Society, 1998, pp. 154–161.

[224] a. Medina, a. Lakhina, I. Matta, and J. Byers, "Brite: an approach to universal topology generation," *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 346–353, 2001.

[225] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski, "Towards Realistic Million-Node Internet Simulations," *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pp. 2129–2135, 1999.

[226] "Scalable simulation framework network models (ssfnet 2.0)," 2004. [Online]. Available: http://www.ssfnet.org/homePage.html

[227] H. Sundani, H. Li, and V. Devabhaktuni, "Wireless sensor network simulators a survey and comparisons," *International Journal Of Computer Networks (IJCN)*, vol. 2, no. 2, pp. 249–265, 2010.

[228] F. Yu and R. Jain, "A survey of wireless sensor network simulation tools," *Washington University in St. Louis, Department of Science and Engineering*, 2011.

[229] G. F. Lucio, M. Paredes-farrera, E. Jammeh, M. Fleury, and M. J. Reed, "Opnet modeler and ns-2 : comparing the accuracy of network simulators for packet- level analysis using a network testbed opnet modeler and ns-2 : Comparing the accuracy of network simulators for packet-level analysis using a network testbed," *WSEAS Transactions on Computers*, vol. 2, no. 3, pp. 700–707, 2003.

[230] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pp. 82–91, 2013.

[231] L. Devroye, "General principles in random variate generation," *Non-Uniform Random Variate Generation*, pp. 27–82, 1986.

[232] C. Boucher, "Sampling random numbers from probability distribution functions | comsol blog," 2016. [Online]. Available: https://www.comsol.com/blogs/sampling-random-numbers-from-probability-distribution-functions/

[233] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 303–314, 1998.

[234] A.-H. Mohsenian-Rad and A. Leon-Garcia, "Optimal residential load control with price prediction in real-time electricity pricing environments," *IEEE Transactions on Smart Grid*, vol. 1, no. 2, pp. 120–133, 2010.

[235] S. Ramchurn, P. Vytelingum, A. Rogers, and N. Jennings, "Agent-based control for decentralised demand side management in the smart grid," *AAMAS '11, Taipei,*, pp. 5–12, 2011.

[236] T. Hubert and S. Grijalva, "Realizing smart grid benefits requires energy optimization algorithms at residential level," in *ISGT 2011*. IEEE, Jan. 2011, pp. 1–8.

[237] B. Neenan and J. Eom, "Price Elasticity of Demand for Electricity : A Primer and Synthesis," Electric Power Research Institute, Tech. Rep. January, 2008.

[238] OpenADR Alliance, "OpenADR." [Online]. Available: http://www.openadr.org/

[239] G. Lilis and O. Van Cutsem, "vMiddlware: a real time building emulation engine," https://gitlab.com/georgekav/ib_backend_virtualization.git, 2017.

Abbreviations and Acronyms

ABS	agent-based simulation
ADR	automated demand response
API	application programming interface
BAS	building automation system
BMS	building management system
CDF	cumulative distribution function
CPS	cyber-physical system
DES	discrete-event simulation
DEVS	discrete event specification
DR	demand response
DSM	demand side management
EMS	energy management system
GNSS	global navigation satellite system
HID	human interface device
HPS	hybrid positioning system
IC	integrated circuit
ICT	information and communication technology
IoT	Internet of Things
IPS	indoor positioning system
LP	load profile
LSB	least significant bit
M2M	machine-to-machine
MCU	microcontroller unit
MoM	message-oriented middleware
MPC	model predictive control
MPU	microprocessor unit
NAT	network address translation
NFC	near field communication
OOP	object-oriented programming
ORM	object-relational mapping
OS	operating system
PDF	probability density function

PLC	power line communication
PMF	probability mass function
pMid	physical middleware
PMU	power measurement unit
PMU	phasor measurement unit
PV	photovoltaics
RDBMS	relational database management system
RFID	radio frequency identification
RPC	remote procedure call
RSSI	received signal strength indicator
RT AM	real-time atomic model
RT CM	real-time coupled model
RTDEVS	real-time discrete event specification
RTT	round-trip time
SB	Smart Building
SG	Smart Grid
SOC	system on a chip
SoC	state of charge
SoM	service-oriented middleware
TSDB	time series database
UML	unified modeling language
uTread	micro-thread
UUID	universally unique identifier
vEngine	virtualization engine
vEntity	virtual entity
vMid	virtualization middleware
vNetwork	virtual network
VPN	virtual private network
VPS	virtual private server
WPS	Wi-Fi-based positioning systems
WSN	wireless sensor network

GEORGIOS LILIS

École Polytechnique Fédérale de Lausanne
Electronics Laboratory · Lausanne, Vaud, CH-1015
+41 766503143 · georgios.lilis@alumni.epfl.ch

EDUCATION

École Polytechnique Fédérale de Lausanne, Switzerland **2017**
Ph.D. in Microsystems & Microelectronics
Dissertation: "A scalable and secure system architecture for smart buildings"

Aristotle University of Thessaloniki, Greece **2012**
Diploma (*joint bachelor's & master's degree*) in Electrical & Computer Engineering
Thesis: Power system dynamic analysis using mixed-signal electronics emulation

SCIENTIFIC INTERESTS

Smart Building • system design, ambient intelligence, demand side management, IoT
Cloud & fog computing • architectures, virtualization, analytics, IaaS, PaaS
Distributed architectures for scalability and fault-tolerance
Software - hardware co-design
Embedded & computer networks
Embedded electronics design
Cybersecurity

TECHNICAL STRENGTHS

Computer Languages	Python, C, C++, Java (SE), JavaScript, HTML5, CSS, VHDL
Full-stack development	Embedded, Android, frontend/backend web development, analytics platforms (Hadoop, Storm, Spark), FPGA, LABVIEW
Protocols	6loWPAN, 802.14.5, LPWAN-LoRaWAN, Powerline, Zwave, 802.11, 802.3, NFC
Embedded	Discrete analog/digital, PCB, microcontrollers (ARM/AVR) microprocessors (MIPS/ARM), SOC
Databases	NoSQL, mySQL, SQLite, HBase
Tools	Git, SVN, Latex, shell script, MATLAB
Misc	Power systems, project management, Agile development

LANGUAGES

Greek	Native proficiency
English	Full professional proficiency (C2)
French	Professional working proficiency (C1)
German	Elementary proficiency

CERTIFICATIONS

PRINCE2® Foundation certificate in Project management – 04117287-01-ESG3
NI® Certified LabVIEW Associate Developer – 100-316-16286

LIST OF PUBLICATIONS

Journal Papers

[1] **G. Lilis**, and M. Kayal "A Secure and Distributed Message Oriented Middleware for Smart Building Applications," in Automation in Construction, 2017

[2] **G. Lilis**, O. Van Cutsem, and M. Kayal A High-Speed Integrated Building Emulation Engine Based on Discrete Event Simulation," in IEEE Transactions on Automation Science and Engineering , 2017 *in revision*

[3] **G. Lilis**, G. Conus, N. Asadi, and M. Kayal, "Towards the next generation of intelligent building: An assessment study of current automation and future IoT based systems with a proposal for transitional design," in Sustainable Cities and Society, 2016

[4] G. Lanz, L. Fabre, **G. Lilis**, T. Kyriakidis, D. Sallin, R. Cherkaoui and M. Kayal, "Calibration of a Mixed-Signal Power Network Transient Stability Analysis Emulator," in International Journal of Microelectronics and Computer Science, vol. 4.4, 2013

Conference Papers

[1] O. Van Cutsem, **G. Lilis**, and M. Kayal, "Automatic Multi-State Load Profile Identification with Application to Energy Disaggregation," in 22nd International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 2017

[2] G. Conus, **G. Lilis**, N. A. Zanjani, and M. Kayal, "Toward Event-Driven Mechanism for Load Profile Generation," in 22nd International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 2017

[3] **G. Lilis**, O. Van Cutsem, and M. Kayal, "Building Virtualization Engine : a Novel Approach Based on Discrete Event Simulation," in 2nd International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP), Krakow, Poland, 2016

[4] G. Conus, **G. Lilis**, N. A. Zanjani, and M. Kayal, "An Event-driven Low Power Electronics for Loads Metering and Control in Smart Buildings," in 2nd International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP), Krakow, Poland, 2016

[5] **G. Lilis**, A. Hoffet, and M. Kayal, "GeoAware : A Hybrid Indoor and Outdoor Localization Agent for Smart Buildings," in 18th Mediterranean Electrotechnical Conference (MELECON), Limassol, Cyprus, 2016

[6] **G. Lilis**, S. Bansal, and M. Kayal, "JouleSense: A simulation based platform for proactive feedback on building occupants' energy use," in 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), Rome, Italy, 2016

[7] **G. Lilis**, G. Conus, and M. Kayal, "A Distributed, Event-driven Building Management Platform on Web Technologies," in 1st International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP), Krakow, Poland, 2015

[8] **G. Lilis**, G. Conus, N. Asadi, and M. Kayal, “Integrating building automation technologies with smart cities. An assessment study of past, current and future interoperable technologies,” in 4th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), Lisbon, Portugal, 2015

[9] N. A. Zanjani, **G. Lilis**, G. Conus, and M. Kayal, “Energy Book for Buildings Occupants Incorporation in energy efficiency of buildings,” in 4th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), Lisbon, Portugal, 2015

[10] **G. Lilis**, T. Kyriakidis, G. Lanz, R. Cherkaoui and M. Kayal, "Pipelined Numerical Integration on Reduced Accuracy Architectures for Power System Transient Simulations," in 16th International Conference on Computer Modelling and Simulation (UKSim-AMSS), Cambridge, UK, 2014

[11] **G. Lilis**, T. Kyriakidis, G. Lanz, R. Cherkaoui, and M. Kayal, “On the Effect of Integration Algorithms on Reduced Accuracy Computational Architectures For the Transient Simulation of Power System Dynamic Phenomena,” in ECESCON 7, Thessaloniki, Greece, 2014

[12] G. Lanz, L. Fabre, and **G. Lilis**, T. Kyriakidis, D. Sallin, R. Cherkaoui and M. Kayal, "Power network transient stability electronics emulator using mixed-signal calibration," in 20th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES), Gdynia, Poland, 2013

[13] T. Kyriakidis, G. Lanz, D. Sallin, **G. Lilis**, L. Fabre, R. Cherkaoui, and M. Kayal, "A mixed-platform framework for Dynamic Stability Assessment," 2013 IEEE Power & Energy Society General Meeting, Vancouver, Canada, 2013