Cape Peninsula
University of Technology

RESIDENTIAL ENERGY EFFICIENCY OVER A PERSUASIVE INTERNET OF THINGS BASED FEEDBACK

By

YANN STEPHEN MANDZA

Thesis submitted in fulfilment of the requirements for the degree.

Master of Engineering in Electrical Engineering

in the Faculty of Engineering and the Built Environment

at the Cape Peninsula University of Technology

Supervisor: Dr. A.K Raji

Bellville
June 2021

DECLARATION

I, Yann Stephen Mandza, declare that the contents of this dissertation/thesis represent my unaided work and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

[illegible]

08 November 2021

Signed **Date**

ABSTRACT

In developing countries today, the population growth and the increasing penetration of higher standard of living electrical appliances in domestic places has resulted in a rapidly increasing residential load. In South Africa, the recent rolling blackouts and electricity price increase only highlighted this reality calling for sustainable measures to reduce the overall consumption and peak load. The cost and effectiveness of energy Utility residential intervention campaigns and the complexities linked to the architectural limitations of Home Energy Management Systems (HEMS) have long restricted grid interventions to the commercial and manufacturing sectors. Nevertheless, the dawn of the smart grid, smart energy meters, low-priced sensors, and embedded devices coupled with Internet-related technologies have paved the way to novel residential energy management interventions involving networking and interaction amongst devices, consumers, and the grid. In this regard, the Internet of Things (IoT), an enabler for intelligent and efficient HEMS, is seeing increasing attention in Home Area Network (HAN) design optimization while mitigating related cost limitations. This work presents the design and implementation of an IoT platform for residential smart-grid applications with the requirement of low cost, interoperability, scalability, and technology availability. The work focuses on the backend complexities of IoT home area networks (HAN) using IoT middleware. Cloud technologies as smart-grid tools augment the quality and services in IoT systems participating reducing the cost and complexities of HEMS. Thus, this work leverages open-source Cloud technologies from Back4App as BaaS to provide consumers and Utilities with a data communication platform for time and space agnostic “mind-changing” consumption feedback, appliance operation control, and Demand-Response Management(DRM) via an Android App. Considering these prerequisites, the platform uses the Open Consortium Foundation(OCF) IoTivity-Lite middleware and implemented different case-study for awareness feedback and demand-side management.

Keywords: IoT, IoTivity, Cloud, Back4App, CoAP, HA, Energy Feedback, HEMS, HAN, Android, DRM.

ACKNOWLEDGEMENTS

I would like to thank all those who contributed in any manner possible to me completing this work.

The Lord God Almighty for the strengths, encouragement, and provision you provided through the different ups and downs to complete this work.

Mr. Ivan, Kholer, former Head of IET department at iThemba LABS for giving a chance, trusting me, and maturing at iThemba LABS and facilitating resource access, and giving directions to shape this work.

Mr. Mike Hogan, Head of Software department at iThemba LABS, for mentoring me in different software technologies that ended up valuable to complete the experimental works for this thesis.

Professor, Paul Papka, Head of Ithemba INIT department for pushing me, giving advice, support, financial assistance, and equipment, and making available iThemba LABS resource that helps me test and gather data throughout this work.

My supervisor Dr. Atanda Raji. Your encouragements, directions, and longsuffering have been central in guiding this works and seeing it to the end.

DEDICATION

The thesis is dedicated to my grandmother Nkama Claire, my first teacher, confidence, and God's love extensions in my life. You taught me to be hard-working, kind, and care for others. Though you are gone, your legacy stays in me and I will keep going further and become a better man to make you proud in the hope of seeing you again when I go the way of all men. With thanksgiving to God, I am honored to have known you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Key Word	Description
AC	Alternating Current
ADC	Analogue to Digital Converter
API	Application Programming Interface
App	software program with graphical interface
CoAP	Constrained Application Protocol
BaaS	Backend-as-a-Service
CRUDN	Create, Retrieve, Update, Delete, and Notify
CT	Current Transformer
DAQ	Data Acquisition Device
D2D	Device-to-Device
D2C	Device-to-Cloud
DR	Demand Response
DSM	Demand Side Management
DRM	Demand Response Management.
ESKOM	Electricity Supply Commission of South Africa
GUI	Graphical User Interface
HEMS	Home Energy Management Systems
HAN	Home Area Network
HTTP	Hypertext Transfer Protocol
Hz	Hertz
ICT	information and communication technologies
IoT	Internet of Things
JSON	JavaScript Object Notation
LAN	Local Area Network
MAC	Media Access Control
MW	Mega Watts

M2M	Machine to Machine communication
P2P	Peer to Peer communication
SG	Smart Grid
SA	South Africa
OS	Operating System.
OCF	Open Connectivity Foundation
OIC	Open Interconnect Consortium
REST	Representational State Transfer
TOU	Time of Use
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WSN	Wireless Network Sensor
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
IEEE	Institute of Electrical and Electronics Engineers
EDR	Enhanced Data Rate
HS	High Speed
MAC	Medium Access Control.
SaaS	Software-as-a-Service
VT	Voltage transformer.

CHAPTER ONE
INTRODUCTION

1.1. Introduction

The growing energy consumption in South Africa causes grave problems to energy supply sustainability. This situation is particularly alarming in households from which originate a significant share of peak loads and energy wastage. The recent rolling blackouts and electricity price increase only highlighted this reality calling for sustainable measures to reduce overall consumption. But in such context, the cost, and complexities of grid interventions in the residential sector has limited the Energy utility initiatives to awareness and educational campaign and flash addresses on digital media to address energy wastage. However, the growing demand emphasized the limitation of these interventions. Therefore, it is required for the grid to extend its technological tools to residential buildings.

HEMS provides consumers with feedback on appliances or equipment operation while providing an automation platform for implementing energy management strategies. However, traditional HEMS designs suffer many constraints and limitations. Indeed, granularity, reusability, scalability, and costrelated limitations have impeded their performance. These have restricted their penetration in domestic space. As a component of the smart grid (SG), HEMS performance depends on its business case being operational. That is, their design will require scalable, reusable, and interoperable backend communication platforms. In other words, HEMS design will need these optimizations for effectivity in the current and future management of energy consumption within the smart grid. However, technology affordability and availability are significant barriers to deploying such intelligent platforms in developing contexts. Nevertheless, the recent advancement in mobile devices, cloud computing, and storage, embedded design highly mitigates these concerns. Thus, the Internet of Things (IoT) is an enabler for cost-effective and flexible communication platform HEMS design.

This thesis explored the applicability of smart-grid IoT enabling technologies in providing an efficient, performant, and cost-effective communication platform for energy management applications, particularly in residential places. In this chapter, a research background is provided. Following are the research objectives and research questions identification. Then, the methodology is defined, and the structure of the thesis is briefly explained.

1.2. Background

Across the world today the electricity consumption is fast reaching the point where resources cannot meet the demand (Smith et al., 2012). Indeed, industrial development and population growth-related demands have increased over the years faster than power systems expansion programs. This situation is highlighted by the increasing stress conditions, resource scarcity, and fossil fuel negative environmental impact (Zhang et al., 2015). The issue is even more pressing in developing countries having significant population growth. In SA, the population growth is closely related to the electricity demand (figure 1.1 below). Indeed, between 1994/5 and 2011/2012, the number of connected households (utility grid-connected) was more than double (4.5 to 9.8 million) in SA.

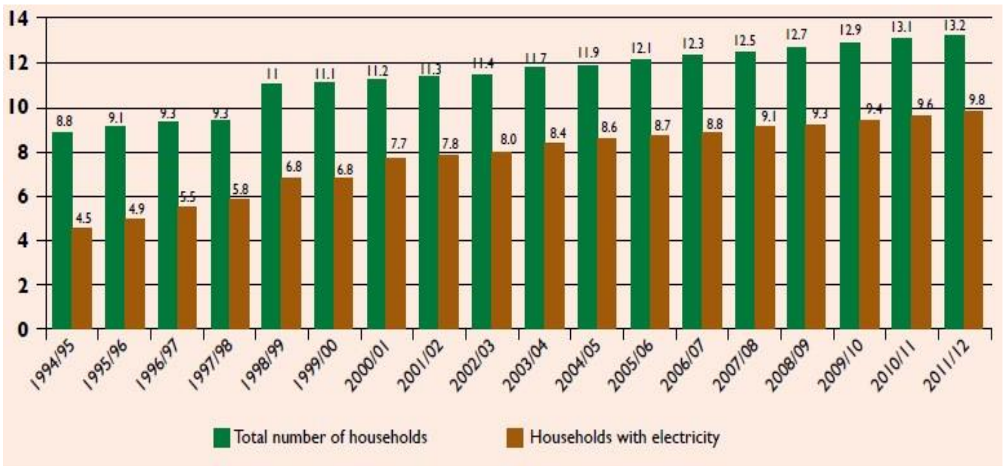


Figure 1.1: Number of Households and Number of Electrified Households (Millions)
(Human Sciences Research Council, 2013)

The increase in this already high demand led to unpleasing energy supply for all related sectors mostly manifested as rolling blackouts due to the load shedding effect. Dealing with load shedding in the winter period (highest seasonal demand), the utility company, Eskom meets a two to four hours shortfall period using an open cycle gas turbine (Deventer, 2014). However, these counteraction measures come with a high cost (operating and maintenance) and significant negative environmental impacts. Considering the aging national grid, there is the need for increased generation and upgrading existing facilities. However, such improvements require significant investment and pose serious environmental threats (Kedar and Somani, 2015). To achieve the needed grid expansion actions, the government and Eskom agreed on a year-based electricity price increase that was between 2008 and 2012(figure 1.2). Such an increase brings the issue of energy affordability for consumers and businesses. Therefore, the efficiency of the power system is seeing a growing interest and is becoming a hot topic amongst researchers (Ki et al., 2016).

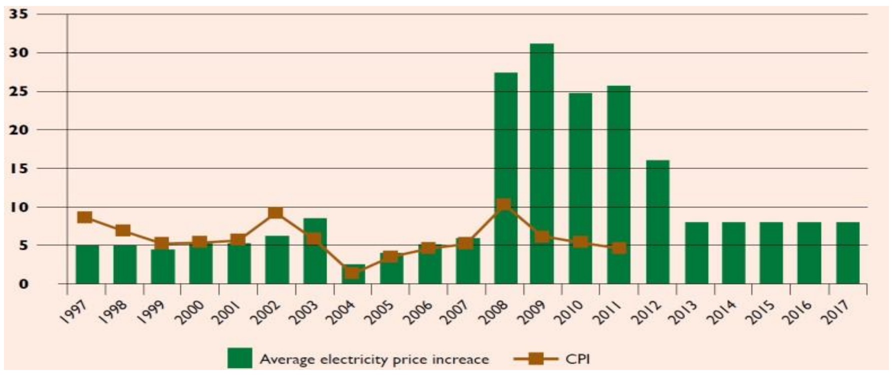


Figure 1.2: Electricity Price Increase in South Africa
(Human Sciences Research Council, 2013)

1.2.1. Residential building consumption

Residential load accounting for around 30-40% of worldwide energy consumption is charged with a significant impact on peak demands(Beligianni et al., 2016). In SA, about 25% (figure 1.3) of energy demand is attributed to residential consumption, a substantial contributor to morning and evening peak results in a national load factor of 72% (Department of Energy, 2012).

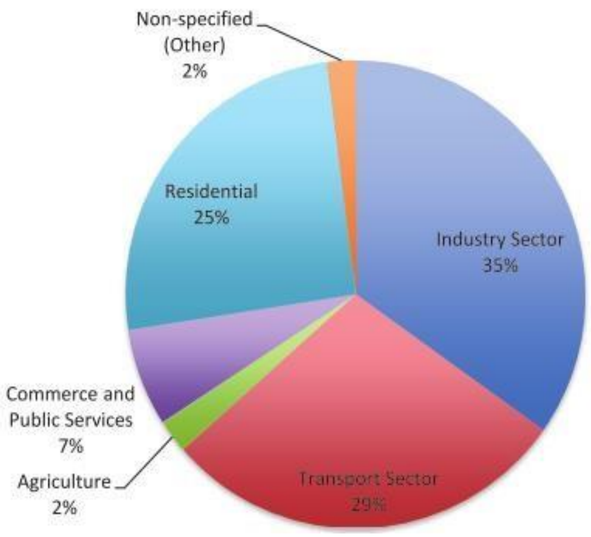


Figure 1.3: Proportion of Total energy consumption by economic sector
(South African Department of Energy, 2016)

Compared to early houses in the late ’80s, modern homes have seen an explosion of high-living conditions appliances penetration which has highly increased residential consumption (figure 1.4). In developing contexts, higher energy usage is generally due to cooling/heating, cooking, lighting, washing, and drying (figure 1.5). According to (Qayyum et al., 2015), these consumption activities (excluding lighting) account for about 70% of appliances in residential buildings. However, residential energy consumption is not only a cause of high consumption appliances. Researchers estimate that about 8% of household energy consumption is attributed to wasteful usage patterns regarding standby (phantom load) appliances (Korsunova, 2010).

Figure 1.4: Evolutions in residential loads and comfort levels
Adapted from (Asare-Bediako, 2019)

Indeed, according to the Department of Energy (2012), compared to the international benchmark for energy poverty (10%), households in SA are spending up to 4% higher of their total household income on energy needs. Energy usage is therefore invisible to most South African households incurring greater misuse.

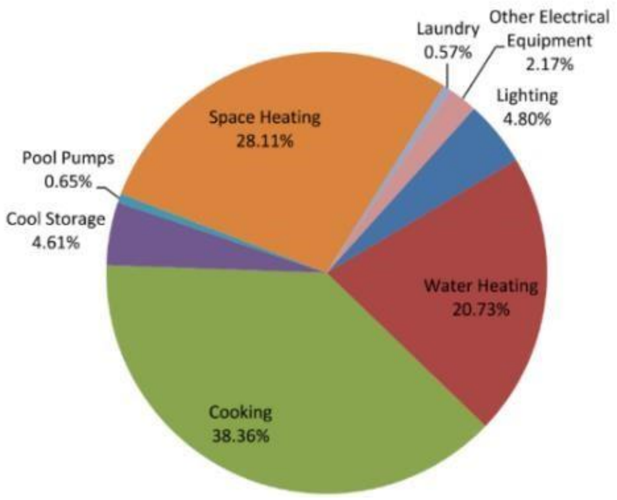


Figure 1.5: Residential Sector Energy End-Use
(South African Department of Energy, 2013)

Though metering devices enable a general perception of consumption, the data obtained is not informative enough and lacks the empowerment capability to curve down consumption. In SA, the power Utility (Eskom) has been engaging in energy-saving advice and educational campaigns with consumers. About the effectiveness of the Eskom approach, Coetzee & Eksteen (2012) writes: “It is highly debatable if this mechanism is effective at all, as most households do not feel that the load restriction applies to them”. Current research in energy

conservation for residential buildings has shown that HEMS in the smart grid (SG) context will bring energy efficiency to households (Kedar and Somani, 2015). Yet, conventional HEMS come with challenges of complexity, affordability, and technology availability. Indeed, their static and fixed network topology limits their reusability and scalability (Kim et al., 2015). Additionally, their usage of proprietary communication protocols restricts the access space and limits consumer products selection, thus increasing interoperability issues and cost (Lee and Lai, 2016). However, the recent emergence of the IoT paradigm in the smart grid context is challenging this reality. IoT devices enable affordable, simplistic, and scalable feedback and automation technologies for home or building energy management (Abdulrahman et al., 2016). IoT facilitates the development of heterogeneous communication platforms around ubiquitous devices that are scalable, affordable, and fit for different use cases
(Jose et al., 2016). Moreover, an IoT-based HEMS can offer “mind changing” feedback bringing energy awareness at the appliance level to motivate and empower individuals to make smart decisions regarding when to turn their electrical equipment on or off (Coetzee and Eksteen, 2012). Furthermore, via open-source tools provided by developing technologies, IoT greatly reduces the cost of HEMS (Korkmaz et al., 2015a).

However, the deployment of IoT systems suffers inherent device interoperability issues (Risteska Stojkoska and Trivodaliev, 2017a). IoT lacks standardization for communication and interaction, resulting in software architecture being unable to scale well to heterogeneous networks (Elfström, 2017). Current research in IoT, advocate for middleware to handle local area network (HAN) complexities. An IoT middleware, a framework running on supported devices offers services regarding semantics gaps, device discovery, and the management of large data (Wang, Hu, Zhou, Zhao, et al., 2015). Recent studies in energy efficiency, leverage cloud computing to extend the capabilities of HEMS. Cloud development gives access to a collection of user-adjustable web services and data storage to mitigate constraints related to resource scarcity and system expansion inherent to residential buildings' automation systems (Korkmaz et al., 2015a). Moreover, cloud computing enables end-users to access and manage HEMS remotely via the internet and mobile devices (Deshpande et al., 2015).

1.3. Research problem and Questions

Energy management platforms for residential buildings need to rely on smart grid enabling technologies for greater efficiency. In this regard, IoT enabling technologies, cloud computing and advances in embedded devices provide an avenue for grid and consumers to effectively participate in the current energysaving effort. In a developing context, smart-grid interventions suffer costrelated implementation issues. Moreover, the initial investment in intelligent appliances required technological tools that can make existing household appliances effectively participate in the energy conservation effort. Thus, the research question of this project, is: “How can IoT enabling technologies within developing context, enhance and facilitate household energy management within the smart grid vision?”

The following question will be guidelines for this research:

1. How can IoT mitigate design complexities and limitations of traditional HEMS?
2. How can IoT mitigate the cost of technology in SG effort for developing context?
3. What backend middleware best handles IoT semantic gaps?
4. How do IoT cloud technologies enhance smart grid effort in the residential sector?
5. How can mobile technologies participate in IoT “mind-changing” feedback to enhance consumer awareness and energy management activities?

1.4. Research Objectives

This work aims to provide energy management interventions in residential places with an effective and performant data communication platform leveraging the smart grid IoT enabling technologies. Thus, the thesis purposed to address the interoperability, scalability, cost, and technology availability of technology constraints of HEMS design and deployment in developing contexts. Lately, a case study illustrating the platform performance in relation to effective real-time feedback, appliances operation control, and a DRM intervention is proposed. In this regard, the thesis objectives are to:

1. Design and implement an IoTivity-based HAN to handle IoT semantic gaps (devices interoperability), increasing HAN device miniaturizations and lowering cost,

2. Design IoTivity smart plugs for interfacing existing home appliances,
3. Optimize and scale the HAN using the cloud as BaaS to simplify the platform backend requirements,
4. Develop an Android-based Energy management App leveraging the cloud BaaS.

1.5. Significance of the research

In developing context, the traditional DR grid for domestic consumers is inexistent mainly due to two causes. First, the great number of domestic units is difficult to handle from the grid side without communication, sensing devices, and effective automation tools. Secondly, compared to their implementation cost, the DR program's effect is negligible. However, as the Smart grid concept is trending amongst research, Utilities are slowly deploying AMI technologies in very constraining conditions in developing contexts. Thus, bringing efficient energy consumption taking advantage of current advances in technology while deploying systems that are scalable to smart grid future AMI are required. The internet and its related technology have been regarded as suitable in this required transition from the traditional grid to the smart grid. This thesis strives while leveraging Internet dependant technology in IoT, cloud computing, embedded design, Web Application to provide a two-way energy management platform mitigating the complexities of HEMS and HAN as well as the cost of their implementation. Thus, this study will be significant in bringing energy consumption literacy, action tacking as well as management at the appliance level, providing a platform that incorporates the existing appliances and residential technological facilities while being interoperable, scalable, and costeffective to greater penetration of the smart grid vision in the residential sector.

1.6. Research Delineation

IoT commands great research interest presenting many design and implementation challenges. Thus, the IoT platform for energy management addresses semantics gaps mainly related to interoperability, scalability issues, and implementation cost. That is, the research primarily focusses on the backend requirements for an IoT platform that offers protocols interoperability, device, and resource management within the HAN and leverage cloud computing to provide energy service on consumer loads. Thus, an IoT middleware as the state-of-the-art tool for ideal IoT applications is used. However, this research does not aim to evaluate the middleware performance in handling the HAN, but this work focusses on the middleware tools that able to satisfy the research primary requirements as to enhance the platform and facilitate its deployment. Security is a primary concern in IoT system deployment. The IoTivity-Lite middleware is used in this work and provides security. However, the different software interactions within it are still in the development stage thus limiting the full implementation of this middleware security layer at this stage of the work. Therefore, the implementation mainly relies on the security layer of the cloud platform as handling security at a higher level of interaction. Additionally, the platform uses login credentials on the energy application to protect the home against unauthorized remote access. To decrease the cost of the IoT platform open-source cloud services and software frameworks provided by emerging technologies in IoT as well low-cost ubiquitous embedded devices are used.

A smartphone Energy App build on the popular Android OS was employ leveraging android technologies pervasiveness in South African households. Although, this work intends to provide a platform for different DRM techniques the performance of these is not part of the evaluation of this platform. Nevertheless, an experimental study was deployed to demonstrate the platform's ability to provide appliance-level feedback, home automation, and a DRM application for a simplistic peak shaving algorithm around common household appliances of resistive type loading.

1.7. Research Methodology

The research aims to both identify and provide a technological solution around IoT middleware, cloud computing, HEMS, HAN, embedded design, and smartphone to develop an effective data communication platform to facilitate greater penetration of the smart grid vision in the residential sector within developing countries. Thus, the research reviews:

- Smart Grid in developing context
- IoT as an enabler for SG and smart home developments
- Home Area Network around IoT technologies
- Embedded design for IoT HAN devices
- Internet of things middleware’s for backend services and semantic gaps
- Cloud Technologies for IoT platforms
- Mobile application as front-end for energy services.

1.8. Outline of the thesis

Following the opening chapter (Chapter One), the remaining of the thesis is ordered as follow:

Chapter Two: This chapter covers the literature on smart grid vision focusing on IoT as an enabler for smart grid penetration in residential load management. Then a review is carried out about the backbone of HEMS or Home Area Network focusing on their architecture, software stacks, and devices within the network. Therefore, this reviewed the current state of the art in embedded design for IoT applications. A review of different IoT middleware implementations as possible solution for IoT semantic gaps is done. Lately a review on cloud computing and related technologies as enablers for IoT-related interventions for energy management is performed.

Chapter Three: This chapter covered specifications that govern the experimental work. Here, the rationale behind the selection of the different tools for the proposed solution was covered as reviewed in chapter 2. Moreover, it covered the new solutions that were developed or adapted to satisfy the research objectives.

Chapter Four: This chapter cover the development of every part of the proposed solution. Firstly, the architecture of the Home Area Network responsible for data communication within the HEMS. Secondly, focus is placed on the design and development of devices that will interface with appliances to provide energy feedback and control interface at the appliance level. In this regard, the research focused on power consumption sensor technologies, electronics control devices as well as firmware development. Thirdly, the backend development for the HAN gateway were handled focusing on its client/server interface both to the local sensor/actuator network and the remote cloud services. Fourthly, cloud service interface is configured to support the application requirements. In this instance, the research concentrates on the cloud storage capacity of for the platform data and communication with the home gateway as well as the remote user front-end. Lastly, attention is place on the development of an Android-based Energy App as a front-end for “mindchanging” feedback and DRM applications.

Chapter Five: This chapter discussed the case study focusing on a DRM scenario used as validating test cases for this research objective as an attempt to answer the research questions. The different scenarios considered in assessing the platform are covered. Simulated loads profiles are defined based on a resistive load setup in a laboratory environment for high-consuming resistive appliances at a fixed RMS consumption. The disaggregated feedback provided both locally and remotely via the Energy App is evaluated. Lastly, the experimental results are presentable in tables and graphs followed by thorough discussion of the results.

Chapter Six: This the study concluding chapter. It includes recommendations and suggests future works.

CHAPTER TWO

SMART GRID AND ENABLING TECHNOLOGIES BACKGROUND

2.1. Introduction

In the traditional grid today, increasing peak load is challenging to the grid infrastructure and often demands from grid management measures involving supplementary power plant procurement, greater rates for consumers, and unwelcome load shedding or even blackouts (Longe et al., 2017). To meet these peak loads, Energy Utility had to grow generation capacity to satisfy the increasing load. Aside from peak capacity plants and storage technologies development, traditional solutions involve satisfying the supply with the required demand using DR to meet the irregular electricity demand (Haider, See and Elmenreich, 2016). In developing context this is mainly done using peak power plants, or direct load control (DLC) techniques avoiding systems overload conditions (Rasheed et al., 2016). Regarding this traditional approach, Haider et al., (2016) argue that these solutions are unsustainable and scarcely affordable as they are unsophisticated countermeasures matching the demand for a limited period as well as increasing environmental-related issues. In this chapter, the focus is on the traditional grid challenges in the South African context. In the latter section, a review is carried out on the relevant technological directions in research for modernizing the traditional grid as well as the stateof-the-art for improving the residential sector demand.

2.2. South African Electric Grid

Aging traditional electric grid lacking intelligent management and situational awareness are now ill-suited to the fast-growing demand for electricity (Paridah et al., 2016b). This situation has placed Energy Utilities under pressure from annual increases in electricity demand coming mostly from residential consumers (Mortaji et al., 2016). Currently, domestic consumption represents about 30~40% of the overall energy use worldwide (Haider et al., 2016). The newest study by the Department of Energy in South Africa shows that on average, households in South Africa spend 14% of their overall monthly earnings on energy necessities a figure above the international standard of 10% for energy poverty. This residential load is regarded as the primary culprit for seasonal and daily peak demand. Following the traditional approach, the Energy Utility in South Africa (Eskom) has reacted to peak demand increase by increasing its power output building new power stations and Adhoc expensive and climate adverse peak power plants, implementing higher tariffs for consumers, and adopting restrictive measures such as undesirable load shedding or even blackouts. Regarding the current investment in South Africa to support the current demand, (Abu-Mahfouz, Olwal, Kurien, Munda, & Djouani, 2015) noted that: “Eskom has R350- billion new-build program (Medupi, Kusile, and Ingula) to fulfil the increase in power demand. However, this program is facing several issues including the fact that it is two years behind the completion schedule, and it incurs significant overruns cost: ongoing labour unrest…”.

In the South African grid, electricity transmission is unidirectional. That is from the generating company Eskom to the consumers (figure 2.2-1 is a close adaptation). Nomusa et al., (2014) describe the interaction between utility and consumer as follow: “The meters in homes are linked to the service provider’s system. A user purchases unit from an agent against their meter number. In turn, recharge occurs when substations calculate supply against the meter number. For those who are not on prepaid electricity, a meter reader collects meter readings manually every month for generation of the bills”. To improve the actual energy interaction the author argues that two-way communication between utility and consumer is most viable as being beneficial to both.

.

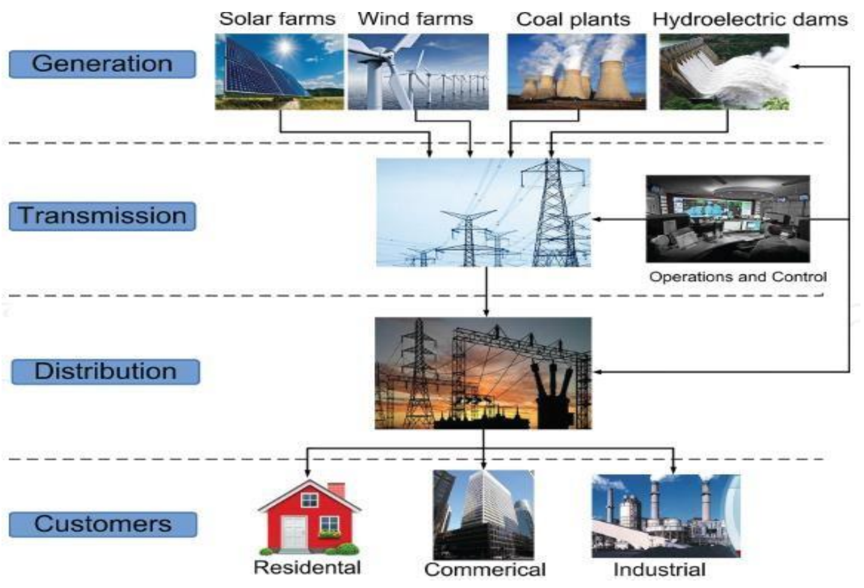


Figure 2.1: Typical Tradition Electric system adapted from (Paridah *et al.*, 2016b)

To effectively reduce consumer peak demand, energy utilities have engaged in different awareness campaigns as well as Demanded Response programs with mitigated results. However, Longe et al., (2017) argue that DRM is essential to peak demand reduction and would benefit both Utility, consumers, and the environment. Hoosain and Paul., (2017) support this viewpoint when stating that: “Traditional investments can be reduced by applying demand response systems” to residential consumption. However, the centralized and unidirectional grid coupled with its inability to accurately profile consumption, the lack of direct and real-time communication between Utilities and consumers limits the effectiveness of the DRM approach in traditional grid systems (Rathi, Raja, Prof, & Rani, 2014). Consequently, with the elimination of past solutions, a new paradigm needs to be adopted (Mortaji et al., 2016). The authors argued that a modern, automated grid to aid in controlling power consumption and increase the effectiveness of the grid via load management is needed to address the current issues. In this regard, the consensus amongst current researchers toward this goal is the smart grid vision.

2.3. The Smart Grid

Traditional grids are dealing with several challenges ranging from aging infrastructure, absence of communication, growing load, and security issues. Moreover, grid interventions to the drastic peak demand increase from Utilities have been supply-side mainly based on Peaker plants and awareness campaigns notwithstanding direct load control. Regarding these measures,
Yardi., (2015) states: “However, these supply-side solutions ignore another attractive alternative which is to slow down or decrease energy consumption through the use of technology to dramatically increase energy efficiency”. Thus, to address the traditional grid these issues, Smart Grid (SG) concept emerged (Lobaccaro et al., 2016). The SG leverages advancement in ICT couple with smart hardware, autonomous and reliable software, for data management alongside a two-way channel amongst consumers and Utilities to consistently and efficiently dispense energy. (Hussain et al., 2018a). As stated by Khan et al., (2019), the SG's main objectives are to expand the consistency, efficiency, and security of power systems. That is, the SG delivers electricity in a controlled and intelligent manner from Utility to active consumers for an efficient and intelligent power system (Risteska Stojkoska and Trivodaliev, 2017a). Moreover, the smart grid concept enables Utilities to efficiently manage peak timings of usage (Qayyum et al., 2015). Consequently, SG add-ons ultimately improve the effectiveness, dependability, and flexibility of the power system (Hussain et al., 2018a).

In traditional power systems, residential energy management has been neglected mainly due to scalability concerns. However, The SG has launched the deployment of smart meters, low-cost sensors, and smart load on top of ICTs in residential units for energy management programs (Lobaccaro et al., 2016). Indeed, the SG has widened the scope of load management to the individual residential unit within the power grid. For instance, Utilities may remotely apply energy management to intentionally reduce peak load.

2.3.1. Smart grid as an Enabler for HEMS

DRM programs, the focus of smart-grid interventions have mainly targeted commercial and big industrial sectors for their large demand reduction to the grid. However, with the increasing portion of residential demand the total grid load, Utilities are showing increasing interest in residential demand response (RDR) (Rastegar, Fotuhi-Firuzabad and Zareipour, 2016). Home Energy Management System (HEMS) is an integral part of a smart grid that can potentially enable DR applications for residential customers (see figure 2.2). A HEMS monitors energy and collects data and manages the operation of domestic electrical appliances by enabling load management techniques according to a pre-determined set of requirements (Blanco-Novoa et al., 2017). Therefore, HEM is critical in realizing demand-side management within the smart grid. HEM provides residential owners the tools to autonomously execute smart load controls via utility signals, customers' preferences, and load priority (Yardi, 2015a). As stated by Kedar and Somani., (2015), by managing consumer demand via two-way communication with the energy market, HEMS optimizes consumer costs.

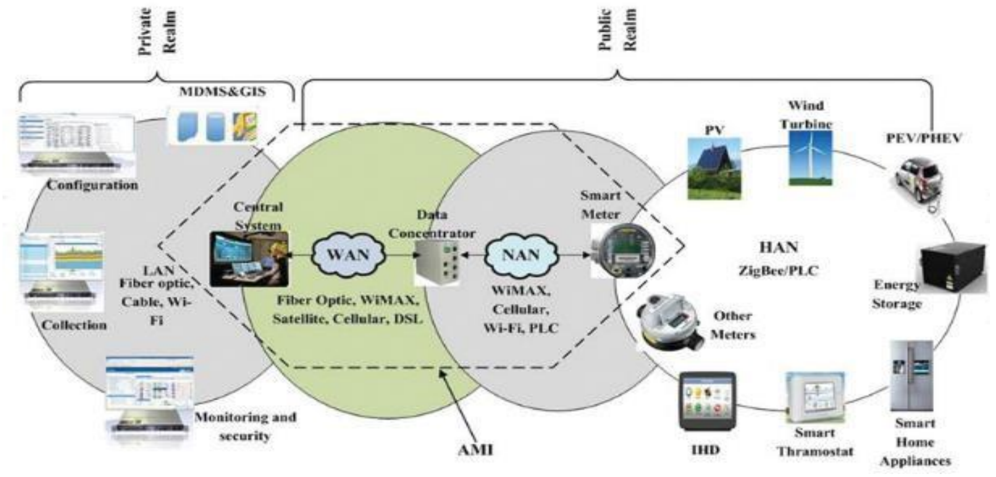


Figure 2.2: HEMS over Smart Grid AMI Modified from (Paridah *et al.*, 2016a)

Residential clients embedded with HEMS have a net reduction of their baseload and peak demand. Aside from DRM application SG based HEMS provides an effective platform for energy literacy via consumption feedback. Indeed, the ability to profile consumption and the advanced analytic and visualization tools proposed by the smart grid enable technologies to allow HEMS to provide an enhanced type of feedback to residential consumers. In this regard, Emeakaroha, Ang, and Yan., (2012), stated that: "energy feedback is a critical foundation for any attempt to reduce energy consumption, and the feedback itself will likely curb energy usage to some extent".

2.3.1.1. Case for Demand Side Management

Electric utilities tend to meet growing consumer energy demand by expanding their generation capacities. These expansions are capital-intensive peak power plants known as "peakers" that are much more costly to operate than baseload power plants (Haider, See and Elmenreich, 2016a). As this strategy results in highly inefficient consumption behaviors and under-utilized power systems, demand-side energy management schemes aiming to optimally match supply with demand have emerged. According to Kailas et al., (2012), Demand-side management (DRM) refers to planning, implementation, and evaluation techniques, including policies and measures designed to either encourage or mandate customers to modify their electricity consumption. DRM interventions impact the behaviour of consumers concerning energy consumption. That is DRM techniques mostly depend on matching current generation figures with load by regulating the appliances' energy usage and enhancing operation at consumer side (Collotta and Pau, 2015). This is mainly accomplished using methods ranging from peak clipping, load shifting, load conservation, valley filling, and load building as illustrated in figure 2.3 below.

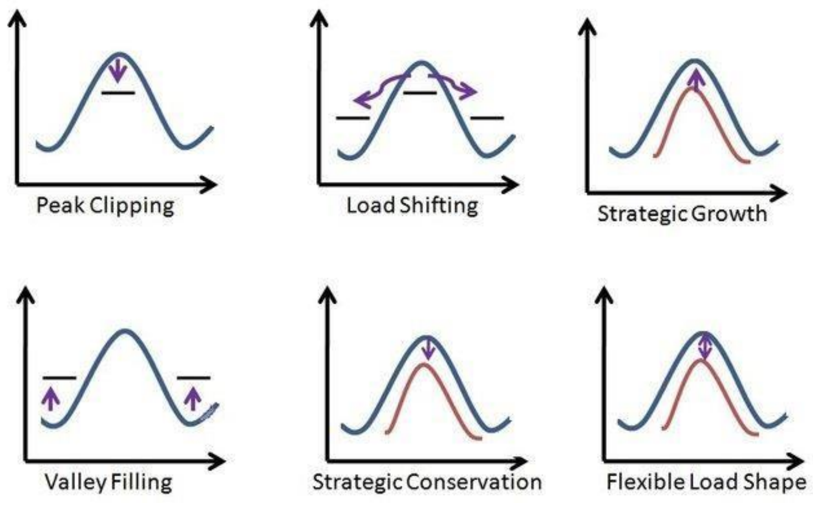


Figure 2.3: DRM Load shaping techniques Adapted from (Kailas, Cecchi and Mukherjee, 2012)

DRM within the smart grid empowers consumers toward intelligent and informed decisions regarding their energy consumption pattern while helping Utilities effectively decrease the peak load when the grid facing higher demand (Khan et al., 2019). Thus, for utilities, DRM means reducing supply costs and avoiding or delaying the need to invest in new capacities (Javor and Janjic, 2017). For the residential customer, it means reduced bills and taking advantage of the financial incentives offered by Utilities (Longe et al., 2017).

Moreover, DRM programs are vital in improving consumers' needs and shaping domestic load for an automated grid (Reka and Ramesh, 2016). The Major benefits of DRM are summed up in table 2.1.

Table 2.1: DRM benefits Adapted from (Kailas, Cecchi and Mukherjee, 2012b)

Customer benefits	**Utility benefits**	**Societal benefits**
Satisfy demand for electricity	Lower cost of service	Conserve resources
Reduce cost	Improve efficiency and flexibility	Reduce environmental impact
Improve service	Reduce capital needs	Protect environment
Improve lifestyle and productivity	Improve customer service	Maximize customer welfare

2.3.1.2. Case for Energy Monitoring (Feedback)

As stated by Suryanarayanan et al., (2013), increasing awareness amongst electricity customers regarding their energy consumption pattern coupled with efficient usage of domestic appliances will bring about effective energy management in domestic places. Studies show that feedback yields great potential in impacting domestic electricity conservation (Vine, Buys, and Morris, 2013). According to Numusa (2012)), using smart meters linked to services provider's systems, south African homes purchase units from agents against their meter number which are recharged after substation calculate supply against the meter number. The authors explained that consumers outside of prepaid electricity, depend on a meter reader that collects readings manually every month for bill generation. This largely un-itemized, non-visual, and infrequent feedback on South African residential has negatively impacted the residential load and encouraging inefficient usage behaviour. This lack of information has become increasingly problematic in South Africa's electric grid suffering from an ever-increasing peak demand from the residential sector.

According to the American Council for an Energy-Efficient Economy, feedback initiatives that make electricity consumption visible to residential users achieve maximum feedback-related savings. Indeed, electricity consumption reduction can be obtained by only providing the consumption profile of appliances to the consumers and accordingly tackling unsustainable behaviour (Collotta and Pau, 2015). Effective feedback according to research is seen as household-specific information on electricity use. Indeed, residential users can achieve optimum feedback-related savings when energy consumption is made visible to them (Lobaccaro, Carlucci and Löfström, 2016). That is disaggregated feedback at the appliance level. Emeakaroha et al., (2012) proposed feedback as direct (real-time), indirect. Direct or real-time feedback is immediate and from a meter or a display monitor and has been found to provide greater energy savings than indirect feedback methods (figure 2.4). Real-time feedback can be more easily customized for individual households. Reviews of direct feedback experiments suggest that this type of feedback interventions yield between 5% and 15% energy savings for the time that they are installed, however, their lasting impacts on behaviour are much less certain (Vine, Buys and Morris, 2013).

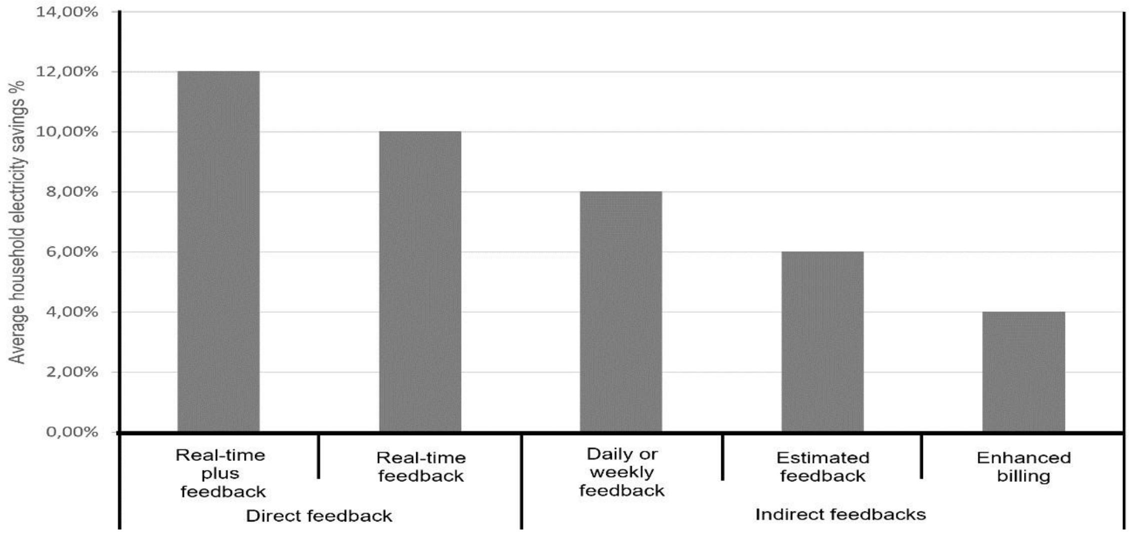


Figure 2.4: Disaggregated feedback type's savings in average Households in the US. Adapted from (Lobaccaro, Carlucci and Löfström, 2016)

Emeakaroha et al., (2012) argued that direct feedback can boost other DR programs, comprising higher user's response and awareness to real-time or time-of-use (TOU) pricing programs and realizing the subsequent profits of load-shifting so impacting peak consumption. Nevertheless, for the power system to provide such feedback, a mixture of beneficial smart grid enabling technologies supported by well-made programs to effectively notify, involve, and persuade consumers(Lobaccaro, Carlucci and Löfström, 2016).

2.3.1.3. The case for Home Area Networks

The smart grid brings new possibilities in residential load efficiency in converting the outdated consumer sites into smart homes via smart appliances and smart meters interconnected as part of a home area network (HAN) (figure 2.5 below). Simply stated, HAN(s) are extensions of the smart grid AMI within a home. Using smart grid technologies, HAN brings monitoring and device control capabilities to energy management (Collotta & Pau, 2015).

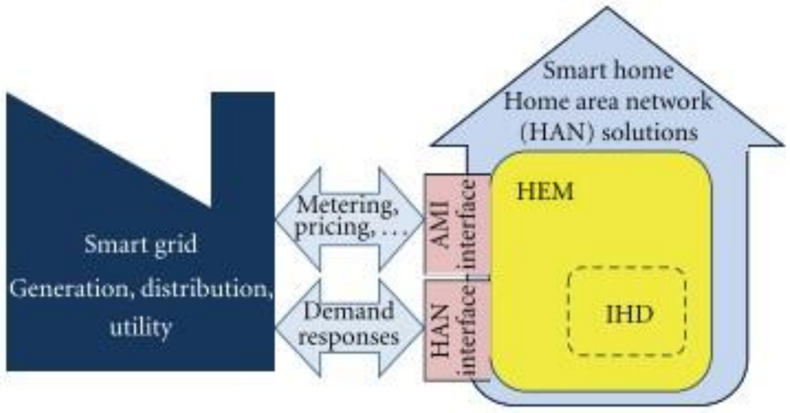


Figure 2.5: Smart Grid Enabling Smart Home and Home Area Networks Adapted from (Kailas, Cecchi and Mukherjee, 2012)

In a smart home, controllable smart appliances interface with smart meters through the HAN. Via the HAN domestic load can be managed and residential grid interfaced with Utility networks. HAN(s) is central for smart grid enable HEM. it provides the data gathering and communication platform used by HEMS to collect information from home appliances that are used to optimize power supply and management (Kim et al., 2015a). The HAN enables smart grid DR interventions at consumers' premises providing a finer control platform for residential consumption efficiency(Zhang *et al.*, 2015). Besides energy management, other functions of HAN in the SG comprise price signals awareness via consumer-centered settings, threshold setting, security monitoring, automatic load control (Longe, Ouahada, Ferreira, & Rimer, 2017)

2.3.1.3.1. HAN Communication and Network Technologies

According to (Kailas, Cecchi and Mukherjee, 2012), HEM is central to green buildings and enables domestic energy consumption monitoring and control usage via smart meters, smart devices and appliances, and smart plugs to provide efficient peak load management(Longe et al., 2017). The different technologies (physical and network layer specification) are generally categorized depending on the communication link into wired and wireless as illustrated in figure 2.6 below. Such technologies provide data communication amongst utility meters and providers, thus building EMS, etc. wired and wireless technologies offer advantages and disadvantages (Table 2.2). Wired technologies provide more security for data communication.

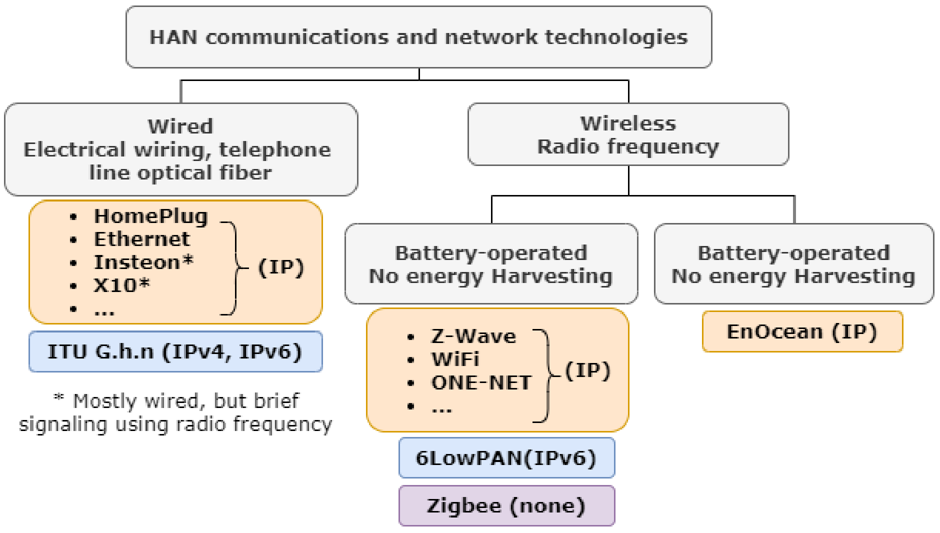
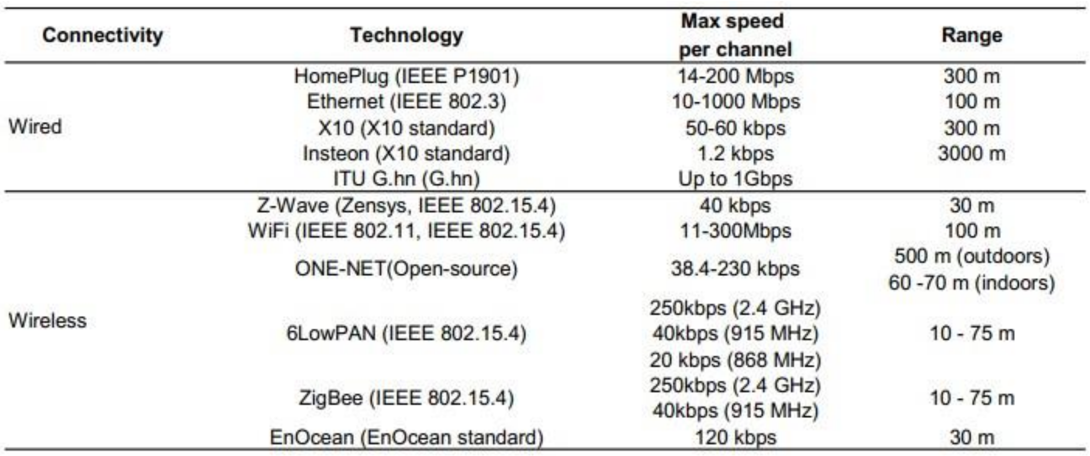


Figure 2.6: Communication and networking technologies for HAN Adapted from (Kailas, Cecchi and Mukherjee, 2012)

However, when it comes to selecting communication technologies, the decision depends on the requirements of the problem. That is, Wired HAN uses transmission channels. That is electronics wiring, telephone lines, unshielded pairs, and/or optical fibre (Kailas, Cecchi and Mukherjee, 2012a). Generally, wired technologies yield a higher deployment cost and are physically prohibitive for any smart grid implementation. Alternatively, wireless technologies reduce equipment and installation costs as well as quicker deployment, extensive access, and superior flexibility which makes wireless preferred to wired technologies (Lobaccaro, Carlucci and Löfström, 2016). Many wireless technologies have been studied and implemented for HAN. However, the ones summarised (Table 2.2) below constituted the main area of focus in research.

Table 2.2: Home Area Network Wireless technologies comparison adapted from (Lobaccaro, Carlucci and Löfström, 2016), (Kailas, Cecchi and Mukherjee, 2012), (Ahmad et al., 2016)



Connectivity	Technology	Max speed per channel	Range
Wired	HomePlug (IEEE P1901)	14-200 Mbps	300 m
	Ethernet (IEEE 802.3)	10-1000 Mbps	100 m
	X10 (X10 standard)	50-60 kbps	300 m
	Insteon (X10 standard)	1.2 kbps	3000 m
	ITU G.hn (G.hn)	Up to 1Gbps	
Wireless	Z-Wave (Zensys, IEEE 802.15.4)	40 kbps	30 m
	WiFi (IEEE 802.11, IEEE 802.15.4)	11-300Mbps	100 m
	ONE-NET(Open-source)	38.4-230 kbps	500 m (outdoors) 60 -70 m (indoors)
	6LowPAN (IEEE 802.15.4)	250kbps (2.4 GHz) 40kbps (915 MHz) 20 kbps (868 MHz)	10 - 75 m
	ZigBee (IEEE 802.15.4)	250kbps (2.4 GHz) 40kbps (915 MHz)	10 - 75 m
	EnOcean (EnOcean standard)	120 kbps	30 m

2.3.1.3.2. HAN Devices

The trend in smart home design today is around embedded devices also known as Smart Home Micro-computers (SHMC). SHMC is a small-sized computer (generally a microcontroller or microprocessor) that can connect to other devices to monitor and control appliance consumption in the smart home system (Lobaccaro, Carlucci and Löfström, 2016). SHMC provides both appliances interface, Smart Grid, and internet servicing to HAN(s). For the latest, SHMC can work as HAN's gateway or Home Energy Controller (HEC). According to (Kailas, Cecchi and Mukherjee, 2012b), the HEC is a networking device that coordinates with the networks within the home and the related ICT protocols for communication with smart appliances. The HEC provides consumers with engaging and simple energy management applications to and control appliances. To energy Utilities, it provides the ability to provision and manage HAN's that monitor, and controls loads as well as very secure end-toend data communications across wired and wireless media and networking protocols. When connected with power sensing and control devices sensor node and WSN mote can be created to monitor and control home appliances. Table 2.3 summarised the strengths and weaknesses of some of the most used embedded controllers in smart home design.

Table 2.3: HAN devices comparison adapted from (Lobaccaro, Carlucci and Löfström, 2016)

Product Name	Main Features	Strenghts	Weaknesses
Arduino	(1) It is an open-source electronics platform equipped with hardware and software; (2) it senses the environment by receiving input from many sensors, and affect its surrondings by controlling lights, motors, and other actuators	(1) High flexibility and compatibility with different kind of sensors; (2) It is intended for anyone making interative project	(1) All these systems require the user to have some technical background and ; electronics basics

Banana Pi	(1) It is a single board computer; (2) It can serve as a platfrom to make mayapplications for different purposes	(1) It targets to be a cheap, small and flexible enough computer for daily life; (2) It is built with ARM Cortex-A7 Dual-core CPU and Mali400MP2 GPU and open source software; (3) Most of common extension accessories includingLCD panel, touch screen, camera module, UART console and GPIO control pin are accessible from Banana Pi on-board connector an header	(2) It also require time to be learned and become expert inassembling and using it. However, many tutorials and detailed information about their assembly and use are free available online; (3) Another barrier is constituted by their commercial price that can also reach thousands of euros
BeagleBone	(1) It is an open hardware micro-computer similar to both Raspberry Pi and Banana Pi; (2) It has an MR cortex-A8 processor. It is equipped with Ubuntu and Android; (3) It is an open hardware, community supported embedded computer for developers and lobbysts	(1) High flexiblity and compatibilitywith many kind of sensors (2) It is intended for anyone making interactive projects	
Raspberry PI	It is a capable credit-card sized computer that allows developing electronics projects	(1) Ability to interact with the external world, and has beenused in a wide array of digital products, from music machines and parent detector to weather stations and tweeting birdhouses with infra-red cameras; (2) Could be used by peaople of all ages; (3) Its challenges is to be used by people of all ages to explore computing and to learn how to program in languages like Scratch and Python and how to minipulate the electronic world around them	

2.3.2. Smart Grid implementations challenges

The implementation of the smart grid suffers from different socio-economical, contextual, and technological factors. That is, implementations of SG technologies on current power systems could be problematic. Firstly, the century-old powerline systems were not designed to meet modern requirements. Secondly, the reconstruction of powerline systems for smart grid applications is costly and tardy. This reality is even more present in developing contexts where technological innovation in this sector has been very slow over time. A commonly cited reason for this slow evolution has been the excessive cost associated with upgrading existing building stock to include “smart” technologies such as network-connected devices (Suryanarayanan et al., 2013). however, the communication architecture of the future smart grid systems is yet to be defined. As a result, lots of challenges and opportunities in the smart grid are defined. A series of challenges in interoperability, scalable internetworking, self-organizing, and security have been identified and discussed (Niyato et al., 2011). Though the implementation issues are generalized throughout the planet, it is important to consider solutions that meet country-specific needs like energy costs, fair billing, electricity thefts, system scalability, and reliability (which are more acute in developing contexts). On the one hand, the open standards-based approach enables seamless integration of the appliances at home to the existing internet infrastructure making remote monitoring, control, and data collection over the web possible (Sahana et al., 2016a). On the other hand, Residential demand response requires advanced metering infrastructure (AMI) which is only now being rolled out. However, in developing context (SA) the idea of the smart grid is still being discussed based on its vision which limits it at present at an infancy state with silo experimentation (no business case effective yet) (Abu-Mahfouz et al., 2015). What Utilities need, therefore, is a way to implement home energy management programs on a widespread basis independently of any AMI deployment plans (Szablya., 2012). Though most limitations to the SG implementation (technological innovations, customer acceptance, utility integration) can at present be overcome, the business case for SG suffers from the current status quo. Therefore, the implementation of the SG is challenging to an energy industry that lacks experience with time-of-use (TOU), critical peak pricing (CPP), and real-time pricing (RTP) rate structure which are at the heart of DRM applications over the SG. Despite consumer acceptance to energy management, Utility still faces implementation related to the maturity of the different standards needed for residential energy management systems (Szablya., 2012). Research agrees that the lack of a robust and inexpensive two-way communications system between the utility and the residential customers represents the main obstacle to residential DR implementations. Though the AMI network within the smart grid will fulfil this need, the business case still needs to be defined and AMI infrastructure deployed by energy utilities.

2.3.3. Smart grid Enabling Technologies

As of the past decade, several emerging technologies and techniques have emerged to support the “smart grid” initiative. Amongst many others, these techniques include advanced metering infrastructure (AMI), two-way communication, and the integration of HAN and HA (Kailas, Cecchi, and Mukherjee, 2012b). The AMI refers to systems of measurement that collection systems that include smart metering at customer premises, network infrastructure between customers and service providers, and data management systems to measure, collect, manage, and analyze the data for further processing (Paridah et al., 2016b). AMI is the backbone of the SG, enabling wider integration of energy and information and ICT (Nomusa, Mudumbe, and Ndwe, 2014). That is, AMI and associated technologies enable real-time, twoway communication between energy suppliers and consumers. Thus, creating a more dynamic control and interaction of the energy flow (Ahmad et al., 2016). Consequently, the extensive deployment of AMI has made available concrete information about user consumption from smart meters (Yardi, 2015b). Furthermore, when coupled with DR, AMI relieves stress from grids at peak times while simultaneously shifting it to non-peak periods providing cheaper cost of electricity production (Thomas, Bansal and Taneja, 2014). Although the business case and the required AMI infrastructure are still to be deployed in developing context, a ubiquitous, reliable, and secure two-way data communications network suitable for DRM that is available to every utility's service area, to the increasing number of homes and is already installed exist. That is the: Internet. Through the internet, greater DRM investment return can be achieved in larger residences.

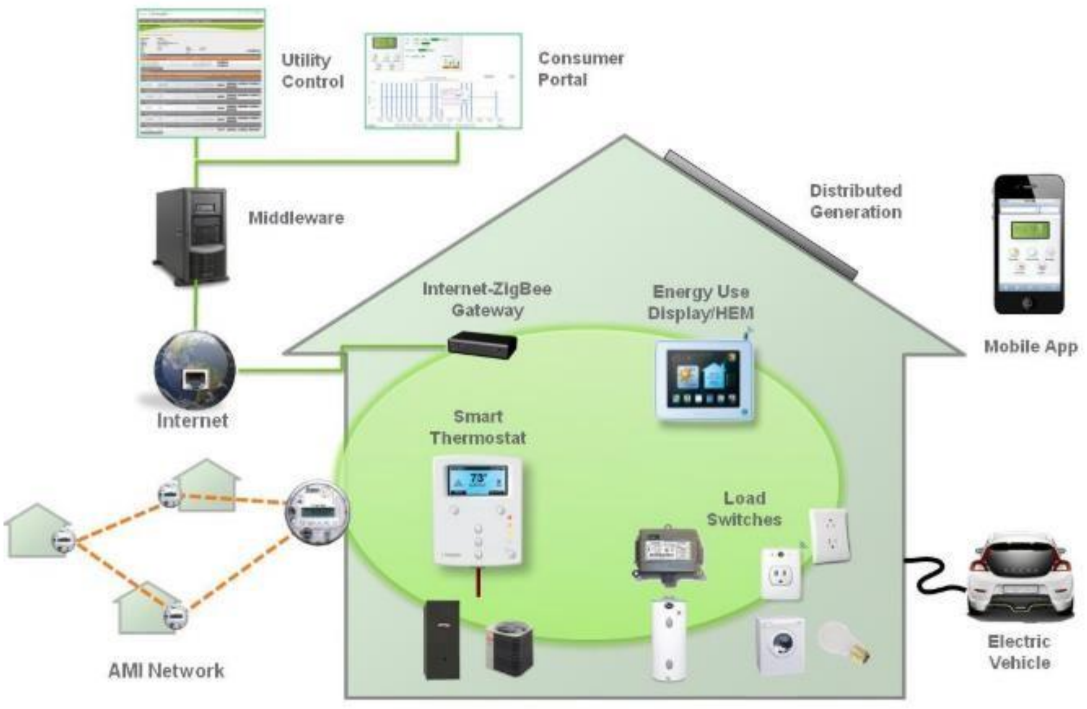


Figure 2.7: Residential Energy Network server over AMI and/or Internet Gateway Adapted from (Szablya., 2012)

The case is also true for developing contexts, where broadband penetration is increasing as DSL, cable modem, 3G/4G cellular communications, and satellite services are expanding (South Africa Connect, 2013). Moreover, rivalry amongst these different options reduces subscription rates. Therefore, for Utilities to leverage the Internet, a dedicated two-way communication platform over a home-installed internet-capable gateway is required (figure 2.7). That is an internet-connected HAN.

2.4. The Internet of Things

Smart devices with communication and computing capacities, going from simple sensor nodes to appliances, and cutting-edge smartphones have become ubiquitous around us. According to Stojkoska et al. (2017): "the heterogeneous network composing of such objects comes under the umbrella of a concept with a fast-growing popularity, referred to as the Internet of Things (IoT)". Simply speaking, the IoT is a novel world connecting nearly every device and appliance to a network. The objects can collaboratively be used to accomplish intricate tasks with a high degree of intelligence and flexibility. In other words, the objects in an IoT network become "smart objects". As depicted in figure 2.8, IoT allows humans and things inter-connection anytime, anywhere, with anything and anybody, preferably via any network, and any services (Razzaque et al., 2016). However, IoT is not a stand-alone paradigm; rather it is an assortment of several technologies working alongside it. This is accomplished via a unified pervasive sensor, data analytics, and data representation via CC as the joining framework (Sethi & Sarangi, 2017).

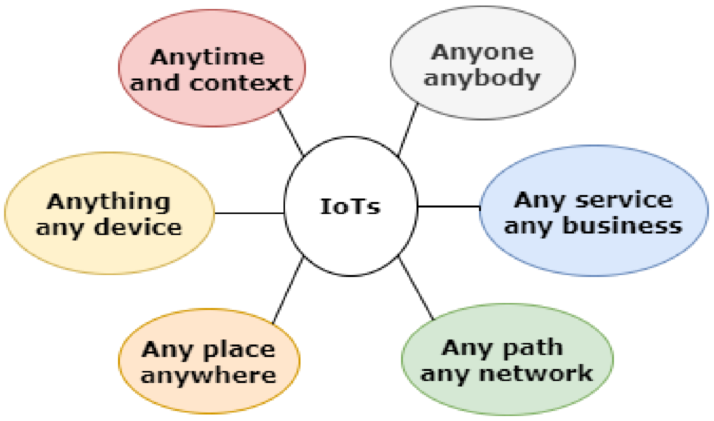


Figure 2.8: IoT Definitions adapted from (Razzaque *et al.*, 2016)

As shown in Figures 2.9, the increasing popularity of IoT will inevitably lead to innovation in several industries, such as smart grid, smart home, intelligent feedback, including new concepts of things technology (Wang, Hu, Zhou, & Zhao, 2015).

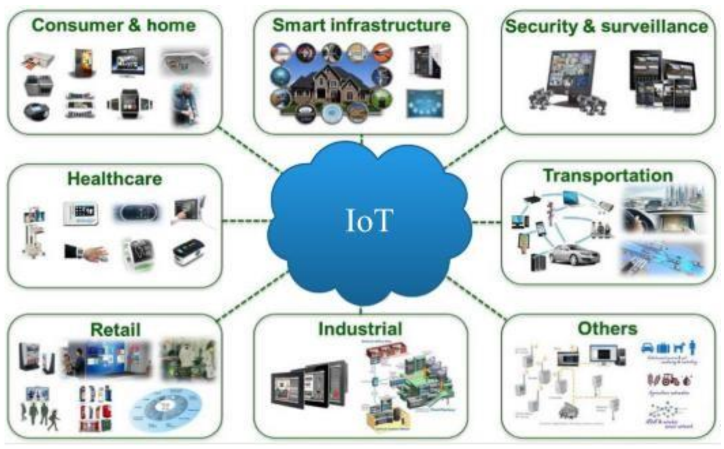


Figure 2.9: Possible IoT applications adapted from (Razzaque *et al.*, 2016)

2.4.1. The case for smart grid infrastructure

The Smart Grid (SG) is the evolving energy system using ICT tools and techniques to make the electric grid perform more efficiently. It possesses demand response capacity to help balance electrical consumption with supply (Naveen et al., 2016). The integration of ICT and power delivery infrastructure into the SG will highly automate the production, distribution, monitoring, and management of electric grids. In this regard, the internet has emerged as a center of focus for smart grid applications enabling non-AMI interventions which offer immediate applications even in developing contexts.

IoT potential is significant for smart grid vision (Wang et al., 2015). Indeed, IoT facilitates electric power management by monitoring and controlling electrical energy production and consumption. consequently, it will lead to significant savings. According to Yao et al., (2016), improving the reliability and sustainability of the SG can be accomplished by easing the perception, aggregation, interaction, and visualization of energy-related information in a real-time and automated fashion. IoT makes the smart grid smarter by integrating intelligent technologies making the SG a network of power systems, telecommunication, and consumer devices (Nomusa, Mudumbe, and Ndwe, 2014).

2.4.2. The case for residential energy efficiency

When coupled with smart metering, IoT has the potential to transform residential places into energy-aware environments. According to Stojkoska et al., (2017), there is an increasing interest in the research community to incorporate the IoT paradigm in the smart-grid concept, particularly for smarthome applications. Madhoo et al., (2015), concurred when stating that Internetconnected devices offer opportunities for better communication and efficient management of energy usage in the residential sector. Aside from M2M interaction, the IoT paradigm enables a synergistic operation between humans and things (human-in-the-loop) offering exciting opportunities to different applications (HEM) (Stankovic and Fellow, 2014). Indeed, HEM systems enable residential energy conservation programs in the smart-grid environment (Yardi, 2015a). In other words, numerous DR strategies can be deployed in the IoTenabled SG to allow active participation at the demand side, thereby improving the mismatch of electricity's demand and supply (Yao, Shen, and Lim, 2016). According to Rathi et al., (2014), the main functionalities of HEM are DRMs, monitoring, and HA by providing economic benefits to both suppliers and consumers via means of intelligent equipment or appliances. Therefore, IoT facilitates Utility DRM programs in the SG context (Coetzee and Eksteen, 2012). By interrelating different technological advancements, IoT enables the development of a platform that facilitates and enhances Utility and the user's effort to save energy.

Loudness and disruption (space and time) are the barriers to technology penetration in the household (Nakajima et al., 2008). In this regard, IoT enhances the concept of "calm-technology". Indeed, using wireless network sensors (WSN), IoT facilitates the implementation of non-disruptive (near invisible) feedback systems although highly interactives. On one hand, IoT enables feedback that is independent of time (real-time and accumulated feedback). On the other hand, feedback is non-disruptive (space requirement) and remote. Therefore, IoT brings energy consumption context information anytime and anywhere. Coetzee & Eksteen (2012), refer to these IoT attributes as "Mind changing feedback".

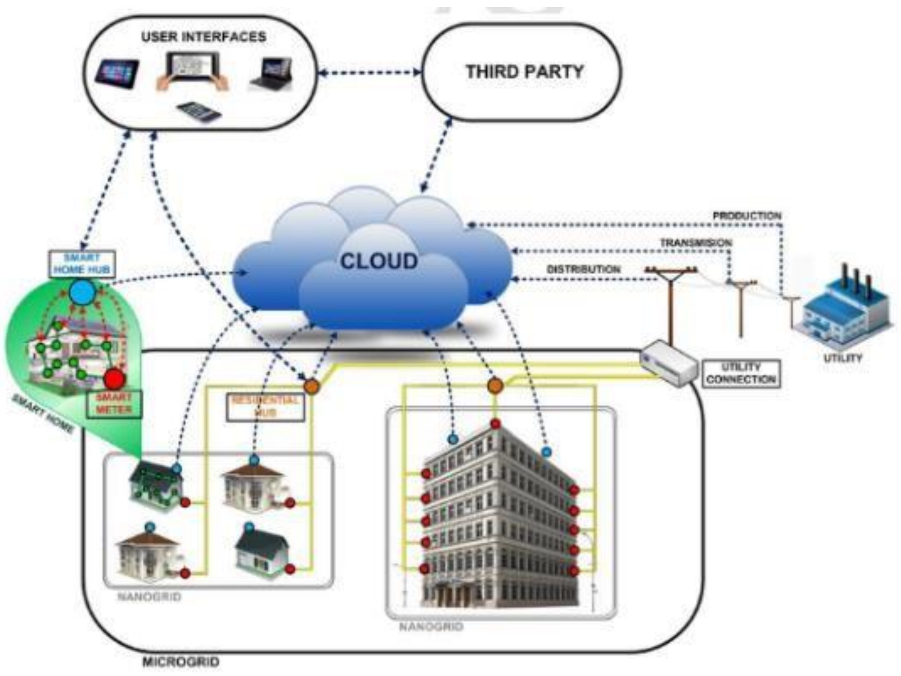


Figure 2.10: IoT Platform for Smart home adapted from (Risteska Stojkoska and Trivodaliev, 2017a)

According to the flesh (2010), of the different value add of IoT to society, "Mind changing feedback" can induce behavioral changes toward energy efficiency. Therefore, Internet-connected appliances that can measure and communicate their energy data will give a deeper insight into those appliances' consumption and facilitate investment toward efficient measures (Coetzee and Eksteen, 2012). In developing contexts, technology affordability and availability, and network connectivity are challenging factors that require attention for successful technological intervention in energy conservation (Vine et al, 2013; Nomusa et al, 2013; Madhoo et al, 2015). Therefore, the question arises concerning the constituent of an IoT-empowered HAN in a developing context. Indeed, HEM systems are either ad hoc or close/uniform for the smart-grid vision. It is therefore critical to implement HEMS architectures that are opened, internetworked. Moreover, these architectures need to support several parallel applications allowing for reliable data/command transfer. According to Li et al., (2015), these limitations can be mitigated, and solutions implemented via the Internet of Things (IoT) paradigm and linked technologies enable the construction of HEMS that are more scalable, reusable, and interoperable.

An effective HAN architecture based on current research focuses on system interoperability (protocols and devices) and scalability (new device easily added or plug-and-play capabilities) (Viswanath, Yuen, Tushar, W.-T. Li, et al., 2016). However, meeting these requirements brings architectural complexities and technology affordability challenges. Nevertheless, recent advancement in ICT, ubiquitous computing, and embedded technologies brings new design possibilities for HAN via IoT (Kim et al., 2015). In this regard, IoT is an enabler for HAN providing intelligent, contextual consumption monitoring and home automation (Collotta and Pau, 2015).

2.4.3. Architecture for IoT

Presently, there is no unique universally agreed-upon architecture for IoT. Therefore, diverse architectures have emerged from different researchers (Sethi and Sarangi, 2017). Viswanath et al., (2017), described a four-layered architecture layer around networking, cloud management, and application as shown in figure 2.11

1. Perception layer senses and gathers data about the environment. This layer is also defined as the device Layer made up of two sub-layers. Things layers enclose sensing devices, actuating devices, smart plugs, and smart utility meters. These smart meters provide data feedback and appliance control. The gateway layer connects elements from the thing layer (Sethi and Sarangi, 2017).
2. The network layer ensures the interconnection between smart things, network devices, inter-connecting the application and the device layer and servers, and provides the transmission and processing of sensor data as well (Sethi and Sarangi, 2017)
3. The application layer delivers targeted application services to the users.

4. The cloud management layer stores data and retrieves information, handles user authentication and data management.

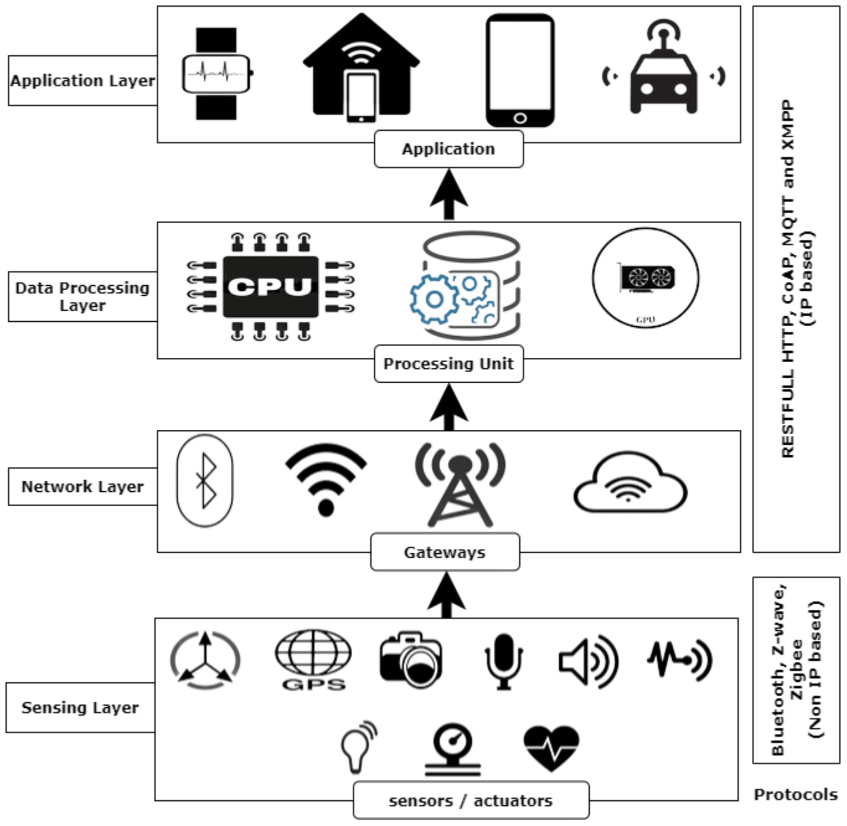


Figure 2.11: IoT communication layers (Viswanath et al., 2016)

2.4.4. IoT enabling technologies

It will be essential for IoT systems to interact and interconnect to offer the "always-promoted, everything-connected" paradigm(Gyrard and Serrano, 2016). However, the IoT vision brings in numerous challenges regarding security, privacy, interoperability, resources and processing constraints, and network capabilities of IoT devices(Minoli, Sohraby and Occhiogrosso, 2017). Figures 2.12 depict the IoT challenges that attract the most IoT research attention.

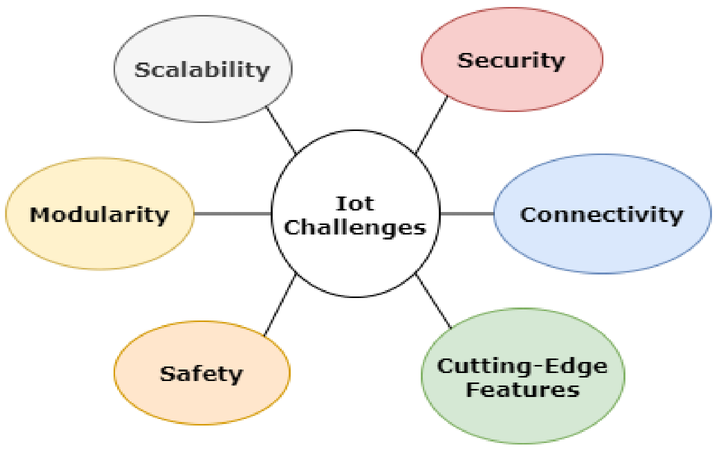


Figure 2.12: IoT challenges adapted from (Abdelsamea, Zorkany and Abdelkader, 2016)

A significant limitation of IoT is the communication and networking of heterogenous devices within Wireless Network Sensors (WSNs). WSNs are challenging in smart grid residential applications introducing several heterogeneous and intelligent devices able to operate, communicate and interact autonomously. Thus, interoperability is critical to successful IoT applications, mainly in smart home applications enabling devices in a network connecting over joint platforms to work together (Viswanath, Yuen, Tushar, W. T. Li, et al., 2016). Therefore, IoT interoperability is vital for smart devices to operate in tandem. Besides interoperability, another major feature of IoT systems is scalability. As an enabler for smart grid applications, it will be critical for IoT networks to be highly scalable to enable a progressive penetration of SG services and smart meters within existing systems. Amongst other software features (mainly at the application level) related to protocol designs should provide space for new devices to be incorporated into the system later while maintaining the firm QoS of existing systems. According to Khan et al. (2016), a key challenge to provide the above-required feature of IoT is standardization. Indeed, IoT must be reliable, easy to use, and secure via the standardization of the different layers of its architecture (Pawlowski, Jara, & Ogorzalek, 2015). Increasing studies among research currently are looking at implementing different standards via integration and optimization of IoT enabling technologies. In Stojkoska et al. (2017), the author specifies five IoT technologies as indispensable for effective IoT solutions, for communication in WSNs, for middleware, for CC, and for application. This interaction is further optimized by incorporating mobile technologies in the application layer of IoT systems.

2.4.5. Sensor and Embedded systems

IoT is a highly Heterogeneous (see figure 2.13) network under whose umbrella interact with embedded electronics devices, sensor, software, and actuator under network connectivity enabling intelligent objects to gather and exchange data (Risteska Stojkoska & Trivodaliev, 2017).

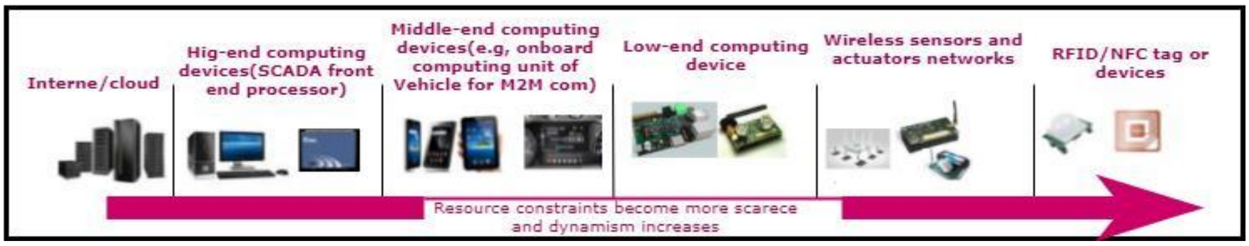

Interne/cloud
Hig-end computing devices(SCADA front end processor)
Middle-end computing devices(e.g, onboard computing unit of Vehicle for M2M com)
Low-end computing device
Wireless sensors and actuators networks
RFID/NFC tag or devices
Resource constraints become more scarece and dynamism increases

Figure 2.13: IoT heterogeneity adapted from (Razzaque *et al.*, 2016)

That is, IoT nodes have actuators, processors, transceivers, and embedded sensors. Information sensed via sensing device is sent to computing units through the Internet. Indeed, the sensor operates as an integrated system or single-purpose CPU embedded into a larger system to control and monitor an environment (Dlodlo et al., 2015). Sensing devices are a critical component of intelligent objects. In IoT systems, these devices are constrained in size, cost, and power effectiveness (Sethi & Sarangi, 2017). In smart-grid applications, current/power transformers have long been used with wide acceptance in different contexts. Khattak et al., (2014), used CT and PT sensors to measure the consumed power from an Energy utility source. According to Ahmad et al., (2016), sensing in the high current application is done via current transformer (CTs) (typically above 100 A). The authors argued that the use of CTs in utility meters is obvious as these precise measurement instruments.

IoT is a novel method to network devices with constrained memory-low computational power, and small power consumption print (Dlodlo et al., 2015). In literature, these intelligent devices are known as embedded systems (ESs). In the past, ESs were manufactured around 8-bit processors with a small memory footprint. But today most embedded devices are built around 32-bit processors with several megabytes of RAM (Buttazzo, 2006). Typically, these smart devices have the following significant properties in common:

- Heterogeneous devices – Embedded devices, sensing devices, and higher-level computing devices to implement heavy-duty jobs. Differences in capacity, features, multivendor products, and application requirements define the heterogeneity amongst IoT devices (Abdelsamea, Zorkany, and Abdelkader, 2016). Figure 2.13 illustrates six different types of IoT devices.
- Resources constraints - IoT devices are built based on memory, weight, and power constraints related to the specific applications. Therefore, embedded applications usually run-on smaller processing devices with restrained memory and computational power. Resource capacity declines (Figure 2.13) from left to right (Razzaque et al., 2016).
- Real-time limitations - Embedded devices interrelate with the outside world promptly. Embedded devices respond to external events while performing computational actions within accurate timing restrictions (Razzaque et al., 2016).
- Power constraints: Network communication consumes much power from battery powered IoT devices. Thus, it is expensive for ES to communicate all the time (Sethi & Sarangi, 2017).

Compared to high-level ES such as Beaglebones, Raspberry Pis, or smartphones, low-level ES usually have less memory, less CPU, consume less power, leverage networks with a reduced amount of data exchange (Roussel, Song and Zendra, 2015). According to Baccelli et al., (2018), Low-level smart devices rely on three core components:

- Micro-Controller (MCU) is a sole piece of hardware comprising a CPU, a few kilobytes of RAM, and Read-Only Memory (ROM), and its register mapped GPIO and memory banks.
- Varied external devices - Actuators, sensing devices, or storage connected to the MCU through GPIO registers to either directly interact with the environment or leverage other sensor and actuator devices.
- Network interfaces (wired or wireless shield) connect devices to the Internet, usually through an energy-efficient transmission technology. These transceivers are system-on-chip (SoC), or devices connected externally through an I/O bus.

2.4.5.1. Sensor networks

WSNs, an emerging area of research consists of smart sensing devices that can communicate through direct radio communication. However, the blending of real-time features with dynamically oriented tasks together with resource and cost constraints generates novel issues to be addressed in WSNs design, at different structural levels (Beligianni et al., 2016). Being made of embedded devices, WSNs face a lot of problems related to limited computational resources, power constraints, low reliability, and higher density of sensor nodes (motes). A mote is a node in a WSN that can perform limited processing, gather sensor data, and communicate with other connected nodes in a network (Dlodlo et al., 2015).

Central to WSN communication in the local network and external world is the gateway. A smart gateway can be employed between underlying networks to provide communication protocol translation within WSN. Although, the Operating systems (OS) are at the heart of the sensor node architecture. That is, the OS is critical in ensuing predictability in the run-mode behavior of the application, thus allowing an offline guarantee of the expected performance. However, typical WSN sensor nodes do not have enough resources to run conventional operating systems (i.e., Linux, BSD, or Windows), thus more compact operating systems have been recently designed (Baccelli et al., 2018).

2.4.5.2. Real-time operating systems (RTOS)

IoT smart things are embedded systems that use real-time operating systems (RTOS) in their development. Abdelsamea et al., (2016), define RTOS as operating systems designed for real-time systems (RTS), or devices having time constraints relative to the system output. The authors point to the capabilities of RTOSs in providing a collection of services. Task management offers multitasking and other significant tools such as Intertask harmonization and communication responsible for information delivered and events shared amongst system jobs. RTOS streamlines ESs development as well as making systems more consistent (Abdelsamea et al., 2016). Mainly RTOS are sought-after features such as scalability, modularity, compactness meaning it boasts an efficient smaller as well as being reliable, predictable, and performant. Lastly, RTOS systems are more efficient than non-RTOS ones when dealing with many tasks or when handling very complex systems (Abdelsamea et al., 2016). Finally, RTOS have nearly continual switching time for several tasks compared to nonRTOS ones (figure 2.14).

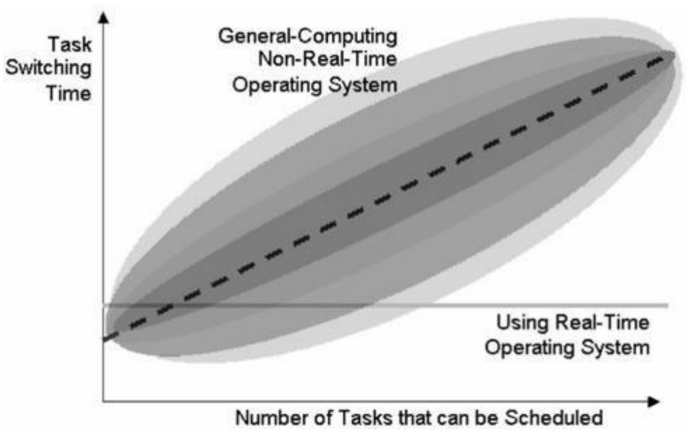


Figure 2.14: RTOS vs. Non-RTOS Systems adapted from (Abdelsamea, Zorkany and Abdelkader, 2016)

As stated by Baccelli et al., (2018), RTOS makes IoT WSNs consistent, effective, as well as predictable. It simplifies their system management. On the integration of IoT and RTOS, many research directions are followed in the literature. RTOS defines many applications system architectures. such integration brings new tendencies in RTOS and IoT integration (Abdelsamea, Zorkany, and Abdelkader, 2016). Figure 2.15depicts an integration that leverages an RTOS in an IoT implementation at the application and network level bridging the two layers using an IoT middleware. Many researchers agreed that such implementations are well suited for smart building applications.

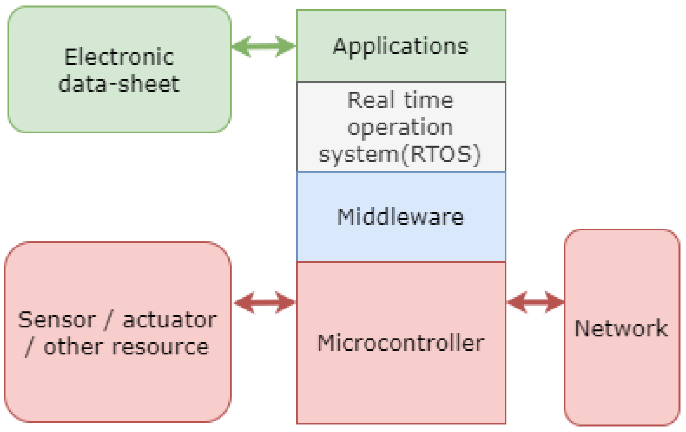

Electronic data-sheet
Applications
Real time operation system(RTOS)
Middleware
Sensor / actuator / other resource
Microcontroller
Network

Figure 2.15: IoT implementation using RTOS adapted from (Abdelsamea, Zorkany and Abdelkader, 2016)

Dedicated Operating Systems for the low-memory devices constituting WSNs in IoT have been designed, distributed, readily accessible for fairly a long time. These range from event-based RTOS for tiny to medium devices (tiny OS) or full-fledge RTOS with kernel and multi-threading support targeting higher-end devices(Baccelli *et al.*, 2018). However, tiny OS is not necessarily software constraints in their implementation but mostly present a low memory footprint, therefore able to run on very low memory devices.

2.4.5.2.1. Contiki

Contiki is a memory-proficient free Operating system for networked embedded devices focusing on power-constrained wireless Internet of Things devices (Sahana et al., 2016). It offers typical OS elements from threads, random number generators, timers, file systems, and others. Contiki offers IPv4 networking support and transport capabilities using TCP/UDP(Kavyashree, 2018). Furthermore, it supports the IPv6 stack in its newest version. ContikiOS has especially been developed for low-powered WSN apps (Sesli & Hacıoğlu, 2017). Below are the main features of the Contiki OS:

- Event-driven Kernel: shrink system size.
- Pre-emptive multi-threading provision.
- Simulator: COOJA
- Written in ‘C’ Language

Contiki runs within constrained memory boundaries. IPv6 network needs less than 10 k RAM and RPL routing needs less than 30 k ROM

(Kavyashree, 2018). It operates on a range of Hardware platforms, and it is easy to port to new hardware. According to Dwivedi et al., (2009), The Contiki kernel, a lightweight event scheduler that posts events to run processes and intermittently calls services polling handlers.

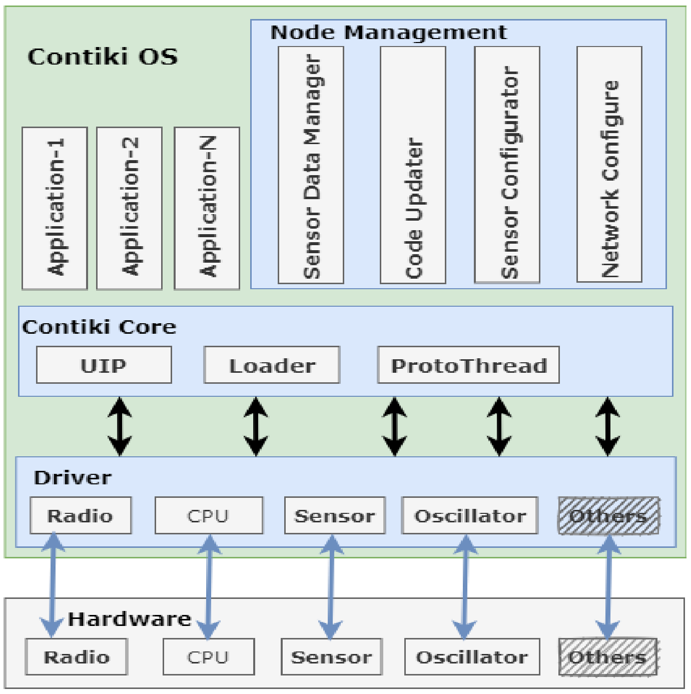


Figure 2.16: Contiki OS architecture adapted from (Dwivedi, Tiwari and Vyas, 2009)

Program implementation is activated either by events posted from the kernel or over-polling mechanism(Kavyashree, 2018). Once an event has been scheduled, the kernel does not pre-empt the execution of the event handler. Contiki kernel provides two types of events: asynchronous and synchronous. It is based on a hybrid model using an event-based model within the kernel and via protothread (an assortment of multi-threaded and event-driven programming mechanisms) providing threading as an application library (Sesli & Hacıoğlu, 2017). The capability to use a unique stack that reduces the needed amount of system constitutes the main advantage of the protothread mechanism. This feature makes it suitable for hardware unable to handle heap memory allocation (Roussel, Song and Zendra, 2015).

2.4.5.2.2. RIOT

RIOT is a free operating system developed on a sectional design assembled around a micro-kernel by a worldwide consortium of developers (Baccelli et al., 2018). RIOT is primarily adapted to low power embedded wireless sensors IoT applications. Open to C/C++ programmers, RIOT offers multitasking and real-time features needing 1.5 kilobytes of RAM only

(Abdelsamea et al., 2016). Indeed, RIOT design was motivated by these goals:

- Minimized memory (ROM, RAM) usage as well reduced power consumption.
- Flexible structures (8-bit to 32-bit MCUs), a wide collection of boards and applications
- curtailed software replication through configurations.
- Code flexibility, across supported hardware (see figure 2.3.4-6).
- Basic programmable software platform.
- Real-time aptitudes.

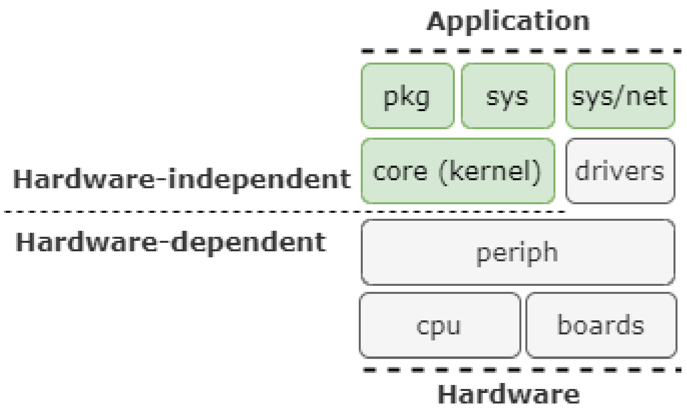


Figure 2.17: RIOT architecture adapted from (Baccelli *et al.*, 2018)

RIOT kernel provides a comprehensive set of features, including multitasking, context switching, event scheduling, inter-process communication (IPC), and harmonization primitives (Baccelli et al., 2018). Other components consisting of the network stack, device drivers, or apps logic are preserved isolated to the kernel. RIOT kernel uses a scheduler built on fixed priorities and pre-emption allowing for soft real-time capabilities. Therefore, RIOT offers a clean way to order tasks and pre-empt management of low-priority jobs dealing with highpriority events. According to Roussel et al., (2015), RIOT beneficial feature compared to other RTOS:

- Efficient, interrupt-driven, tickles micro-kernel
- Priority-aware task scheduler, offering pre-emptive multi-threading
- Extremely effective use of concurrent hardware timers offering the ability to schedule actions with high granularity. This offers the ability to schedule events at 32 microseconds resolution.
- RIOT stack code is written in C. But, contrary to Contiki, there are no restrictions on available constructs.
- Clean and segmental design, making development easier and more productive.

The authors above argue that, amongst the feature here above, the first three make RIOT a mature real-time operating system. However, RIOT present a major drawback compared to Contiki in that it demands a higher memory footprint (Roussel, Song, and Zendra, 2015). This can be explained by the fact that RIOT Operating System has traditionally been developed for ARM hardware and only recently been ported on more recent MCU architectures see (figure 2.18 below).

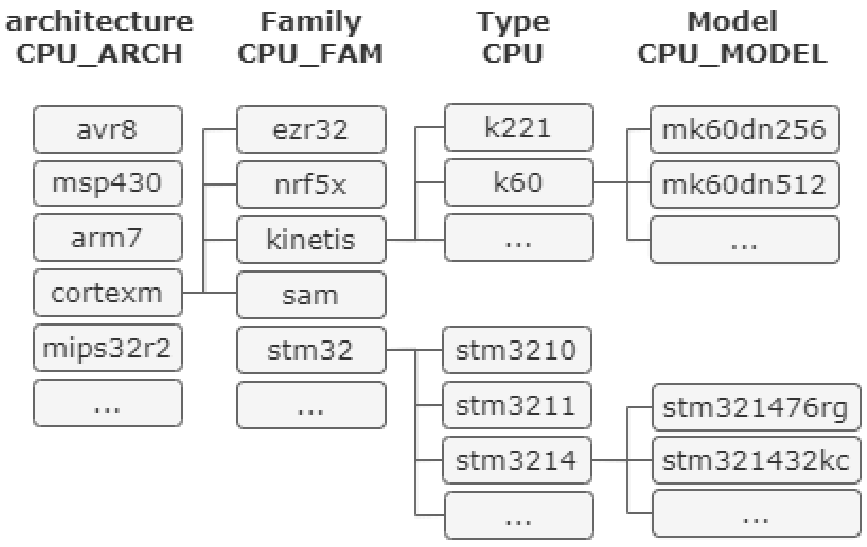


Figure 2.18: RIOT supported CPUs (Baccelli *et al.*, 2018)

2.4.5.2.3. FreeRTOS

FreeRTOS is a real-time kernel or scheduler developed for memory constraints to run on a low-resources device. Usually a kernel byte-code image ranges from 4 kB to 9 kB (Lazic, 2016). FreeRTOS is preferably wellmatched for embedded real-time applications using MCU or low-memory microprocessors (Barry, 2016). FreeRTOS source code is written in assembly language and C. It is available on most embedded systems architectures as a library to create real-time, multi-tasking, embedded applications (Déharbe, Galvão, and Moreira, 2009). FreeRTOS has many developmental benefits. It is open source, well documented as well as boasts a large community. Besides real-time events the main features of FreeRTOS are:

Task management - In multi-threading application tasks serve as computational units. In FreeRTOS, the task hold state alludes to its current mode of activity. That is, a task is either running, in a ready state, or blocked.

FreeRTOS uses priority (an integer value set a compilation time) to schedule task execution from the highest priority setting. (Lazic, 2016).

- Inter-task Interaction and synchronization - FreeRTOS makes use of queues, Semaphores (derivative of queues held when the queues are empty otherwise released when the queues are filled). Additionally, counting Semaphore control entrees to resources to a determined number of running jobs as well as mutexes with priority inheritance to avoid the priority inversion problem (Déharbe, Galvão, and Moreira, 2009).
- Memory management - FreeRTOS makes use of three memory provisioning models. Static memory allocation for each created task has the side effect of memory space wastage. The medium model dynamically provisions memory space and uses a best-fit model to find unoccupied space in memory. The hardest models use a userdefined algorithm for task particular requirements (Lazic, 2016).

2.4.6. Communications and protocols

The rapid growth of IoT is bringing a ubiquity of smart devices that are connecting to the internet over a multiplicity of communication channels and protocols (Figure 2.19). Sethi & Sarangi., (2017), list the issue below as the main challenges for communication in IoT networks:

- *Addressing and identification* - A large addressing space is needed as millions of “things” are going to be connected and a distinctive address to every smart node will be needed.
- *Power-constrained communication* - Interaction between devices is a power-consuming task. That is, low power communication solutions are required.
- *Routing protocol with low-memory print* based on efficient communication patterns.
- Non-lossy and fast communication, and Flexible and Mobile smart devices

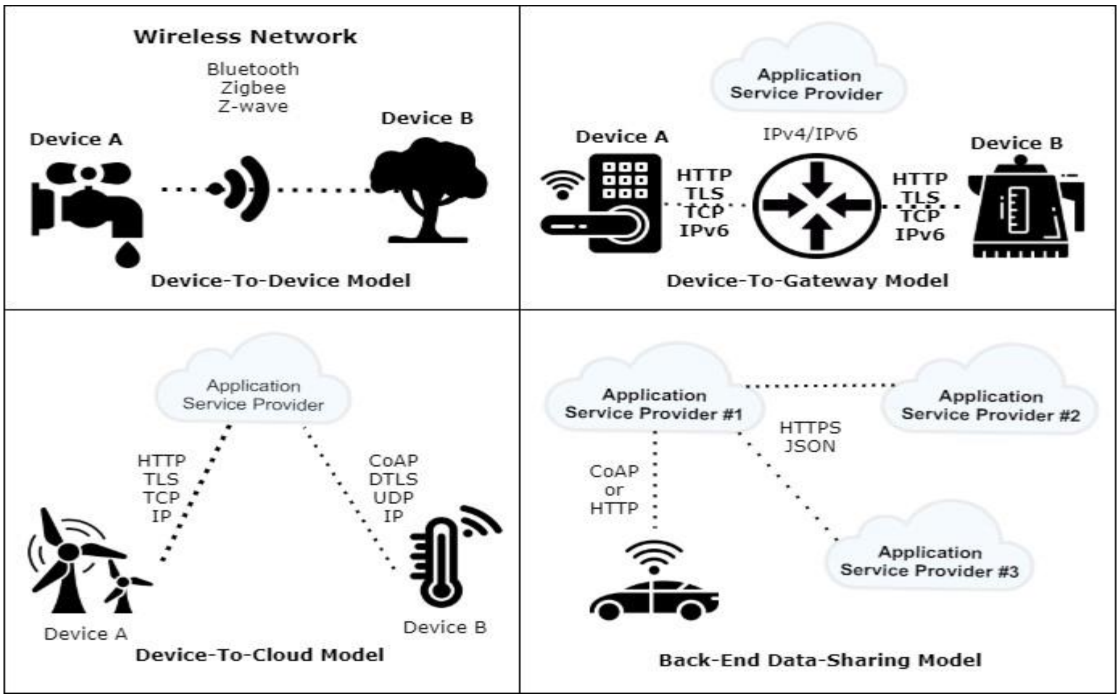


Figure 2.19: Overview of communication protocols and models for IoT Adapted from (Elfström, 2017)

Inherent resource constraints of IoT devices, heterogeneity, and scalability are limiting factors for the deployment of IoT networks. According to (Ko et al., 2011), The limiting factor to the development of smart home solutions lies in the complexities and cost associated with the integration of intelligent devices. In this regard, smart device-producing companies around the world are tackling the interoperability challenges to ensure simple and seamless integration of new devices to the existing communication infrastructure, mainly the internet (Risteska Stojkoska & Trivodaliev, 2017). As the number of IoT devices with communication-intensive architecture increases, Viswanath et al., (2016) argued that standardization of software protocols is critical in enabling devices from different manufactures and features to effectively communicate. To support this vision, the Internet engineering task force (IETF) working groups have recently undertaken the definition of regular protocols at a different level of the network stack to simplify Internet solutions translation through the IoT protocol stack (Haider, See, & Elmenreich, 2016). Figure 2.20 displays the standard IoT protocol stack and the general Internet stack.

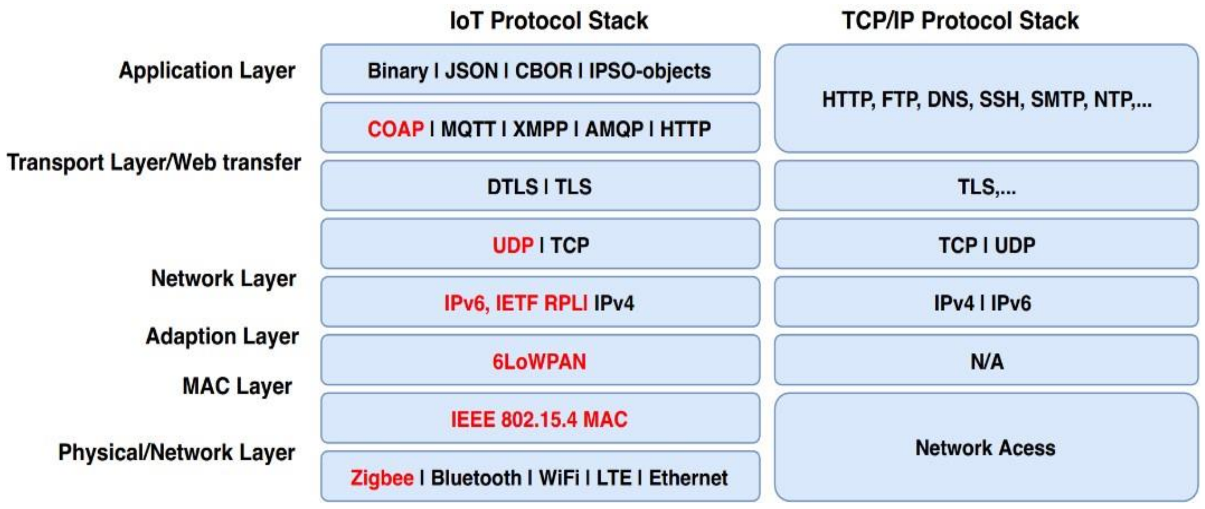


Figure 2.20: Standard Internet and IoT protocol stack; in red are the ITEF standards modified from (Haider, See and Elmenreich, 2016) and (Elfström, 2017)

2.4.6.1. IoT protocol stack

The IoT protocol stack parallels the current OSI reference protocol layers to define an internet communication structure for IoT devices targeting the different limitations of smart devices and complexities of low-power private networks.

2.4.6.1.1. Network layer

The internet protocol (IP) stack is typically used in IoT networks to interconnect smart things in HAN(s) as well as connecting to the Internet. IP is very complex and demands a large amount of power and memory from the connecting devices. The IoT protocol stack proposes non-IP networks, which consume less power, and connect to the Internet using smart gateways (Sethi and Sarangi, 2017). IPv6 is considered the best protocol for communication in the IoT because of its scalability and stability(Kalmeshwar and K S, 2017). IPv6 over Low Power Wireless Personal Area Networks(6LoWPAN), is a very popular standard for wireless communication enabling communication using IPv6 over the IEEE 802.15.4 protocol(Haider, See and Elmenreich, 2016). Indeed, 6LoWPANdevices can communicate with all other IP-based devices on the Internet via a gateway (WIFI or Ethernet), which also has protocol support for conversion between IPv4 and IPv6 as today’s deployed Internet is mostly IPv4 (Sethi and Sarangi, 2017).

2.4.6.1.2. Transport layer

The transmission control protocol (TCP) is not suitable for communication in resource-constrained and low-power environments as it has a large overhead because it is a connection-oriented protocol. The User Datagram Protocol (UDP) is instead preferred because it is a connectionless protocol and has low overhead(Kalmeshwar and K S, 2017).

2.4.6.1.3. Application layer

Inside the IoT protocol stack, the application layer ensures the formatting and presentation of data. On the Internet, this layer is centered on HTTP which is unsuitable for low-resource environments. indeed, HTTP is verbose and incurs large parsing overheads (Sethi and Sarangi, 2017). Shown below is a division of potential IoT standards and technologies applicable for HEMS.

Constrained Application Protocol – CoAP, a dedicated web transmission protocol for low-resources devices and networks (Sahana et al., 2016). CoAP is an alternative to HTTP and offers a request/response interface model between use-case endpoints. it provisions integrated service and resource finding and comprises important web notions (Viswanath, Yuen, Tushar, W. T. Li, et al., 2016). However, unlike HTTP, CoAP provides multicast support as well as using the EXI (Efficient XML Interchanges) binary data formatting is much more efficient regarding space management than plain text HTML/XML (Sahana et al., 2016). To effectively suit constrained-bandwidth communication and constrained computational power device CoAP that is simpler and low-latency as well as being connectionless compared to TCP uses the UDP protocol (Lin and N. W. Bergmann, 2016). Being Stateless, CoAP is structured around the clientserver architecture model. The model is based on the REST architecture which assigns to each resource on a server a unique uniform resource identifier (URI). Using GET, POST, PUT or DELETE methods, a client can request a server to access a resource (see semantic in table 2.4 below) (Madhoo et al., 2015). For security, CoAP employs Datagram Transport Layer Security (UDP-based) as its security protocol (Lin and N. W.

Bergmann, 2016).

Table 2.4: Verbs in CoAP according to RFC 7252 adapted from(Soto, 2017)

Verb	Description
GET	Retrieves a representation of the resources identified by the URI.
POST	Request that the representation enclosed in the request be processed. It usually results in new resource being created or the target resource being updated.
PUT	Request that the resource identified by the request URI be updated or created with enclosed representation.
DELETE	Request that the resource identified by the request URI be deleted.

Message queue telemetry transport: MQTT is a lightweight publish/subscribe messaging protocol, intended for low-memory and low bandwidth, unpredictable networks running over TCP. Subscription’s coordination and authentication jobs used by clients are handled by brokers publishing or subscribing to a topic (Viswanath, Yuen, Tushar, W. T. Li, et al., 2016). As a lightweight protocol, MQTT is suitable for IoT applications. However, being based on TCP, the range of IoT applications using MQTT is limited. Moreover, its messaging content is text-based which increases its overhead (Sahadevan et al., 2017).

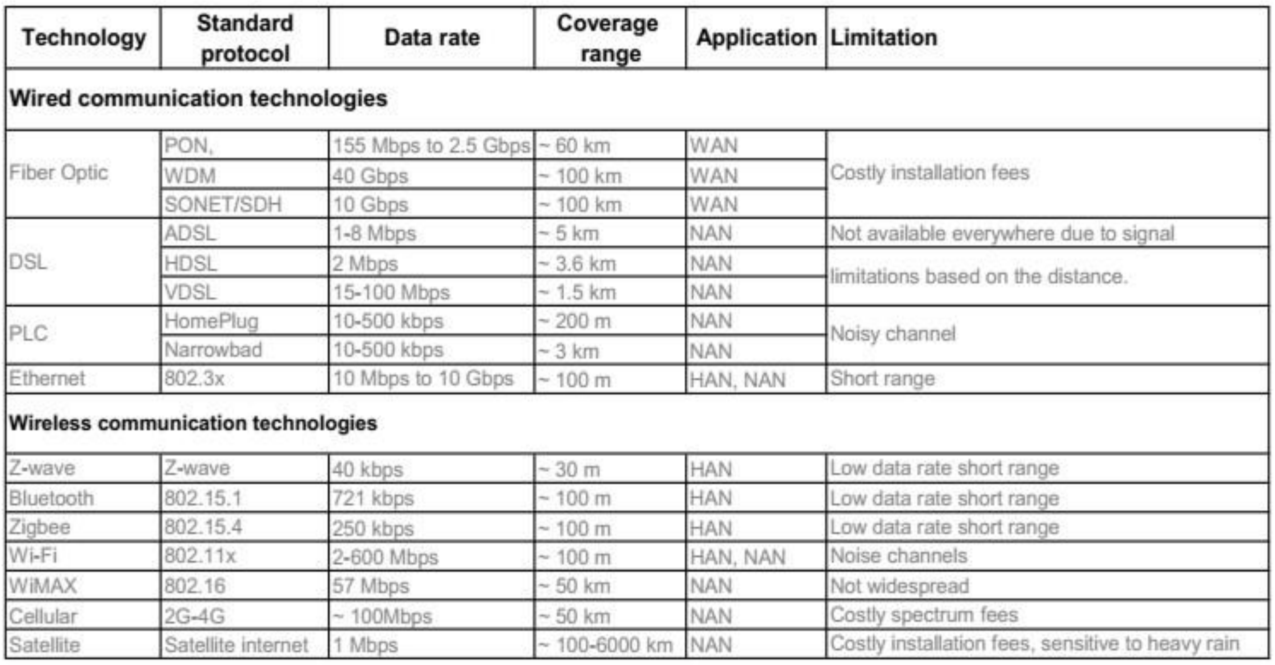
RESTful HTTP: compared to protocol or standards, REST is an architectural model (Piyare, 2013). RESTful uses the REST principles with its implementation resting on the HTTP protocol. Therefore, HTTP implemented on REST is light, has basic request presentation, and is easy to implement. As stated by Viswanath et al., (2016), REST is suitable for asynchronous communication, and end-users’ application which can leverage HTTPS to enhance privacy.

2.4.6.1.4. Datalink layer

Local communication methods are needed for sensing and actuating devices distributed at different deployment locations (Haider, See and Elmenreich, 2016b). According to Ahmad et al., (2016), the successful development of IoTcentric HAN platforms relies on communication infrastructure ability to satisfy the complex requirements of such networks. However, the complexity of IoT HAN’s brings difficulties in selecting appropriate communication architecture and mediums because many parameters and requirements should be considered depending on the applications.

IEEE 802.15.4 specifies the physical layer and media access control for lowrate wireless personal area networks (WPAN). It is the foundation for several higher-level specifications there are three kinds of common network transmission ways: wireless network (Wi-Fi, Bluetooth, etc.), wired network (Ethernet cable or fibre cable), mobile communication network (CDMA, GSM) (Li et al., 2015). In Table 4 are the various communication technologies appropriate to smart grid applications in terms of data rate, coverage distance, and limitation (Haider, See and Elmenreich, 2016).

Table 2.5: Summary of standard IoT communications technologies.
adapted from (Haider, See and Elmenreich, 2016)



Technology	Standard protocol	Data rate	Coverage range	Application	Limitation
Wired communication technologies					
Fiber Optic	PON,	155 Mbps to 2.5 Gbps	~ 60 km	WAN	Costly installation fees
	WDM	40 Gbps	~ 100 km	WAN	
	SONET/SDH	10 Gbps	~ 100 km	WAN	
DSL	ADSL	1-8 Mbps	~ 5 km	NAN	Not available everywhere due to signal
	HDSL	2 Mbps	~ 3.6 km	NAN	limitations based on the distance.
	VDSL	15-100 Mbps	~ 1.5 km	NAN	
PLC	HomePlug	10-500 kbps	~ 200 m	NAN	Noisy channel
	Narrowbad	10-500 kbps	~ 3 km	NAN	
Ethernet	802.3x	10 Mbps to 10 Gbps	~ 100 m	HAN, NAN	Short range
Wireless communication technologies					
Z-wave	Z-wave	40 kbps	~ 30 m	HAN	Low data rate short range
Bluetooth	802.15.1	721 kbps	~ 100 m	HAN	Low data rate short range
Zigbee	802.15.4	250 kbps	~ 100 m	HAN	Low data rate short range
Wi-Fi	802.11x	2-600 Mbps	~ 100 m	HAN, NAN	Noise channels
WiMAX	802.16	57 Mbps	~ 50 km	NAN	Not widespread
Cellular	2G-4G	~ 100Mbps	~ 50 km	NAN	Costly spectrum fees
Satellite	Satellite internet	1 Mbps	~ 100-6000 km	NAN	Costly installation fees, sensitive to heavy rain

2.4.7. Middleware for IoT smart grid applications

For information fusion and ease the process of IoT application development, middleware is highly emphasized and are becoming a widely employed approach (Hu et al., 2016). According to Lin & Bergmann., (2016), a middleware is a software layer that sits between the low-level layer of devices and the highlevel application layer. It consists of a collection of enabling underlying software allowing several processes to connect through a network. The authors summarized the relationship as three visions of the IoT. That is, things-oriented, semantic-oriented, and Internet-oriented. According to its three characteristics, middleware in the IoT shall address internet and things issues, handle semantics gap, context awareness, device discovery, manage devices resources, scalability, big data, and privacy (Elfström, 2017). Via middleware, different IoT communication models can be integrated for richer IoT applications. However, devices in IoT networks are resource constrained. That is, direct internet connections of all devices are unrealistic as such connections are costly regarding computation power, bandwidth usage, and equipment cost (Christen et al., 2014). This makes persistent internet connectivity on devices challenging when trying to maintain device miniaturization in IoT networks. Due to such difficulties, IoT solutions need to support different types of devices with different resource limitations and capabilities. Figure 2.21 categorizes IoT devices based on cost, memory, communication as well as computational power. To achieve this requirement, Razzaque et al., (2016), argued that an ideal IoT middleware solution should be able to take advantage and adapt to these different types of devices so to make the solution more efficient and effective. Perera et al., (2014), further pointed to device granularity (as support for lower-level categories) as essential for an ideal middleware. Indeed, middleware solutions designed specifically for resource-constrained devices are critical to enabling IoT in applications such as smart grid

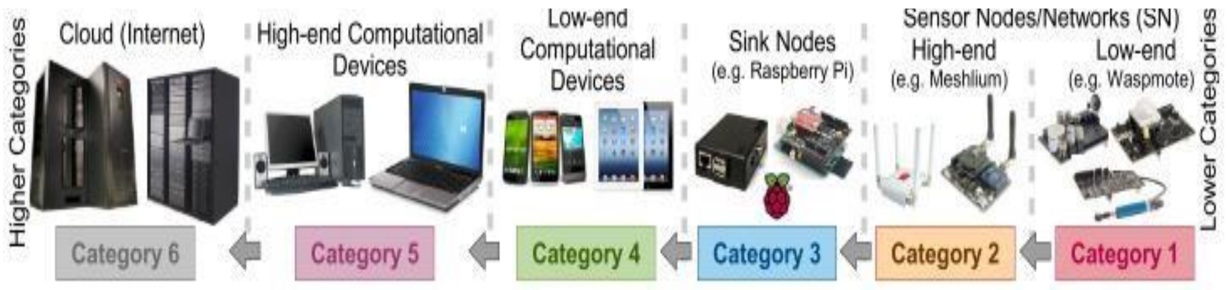
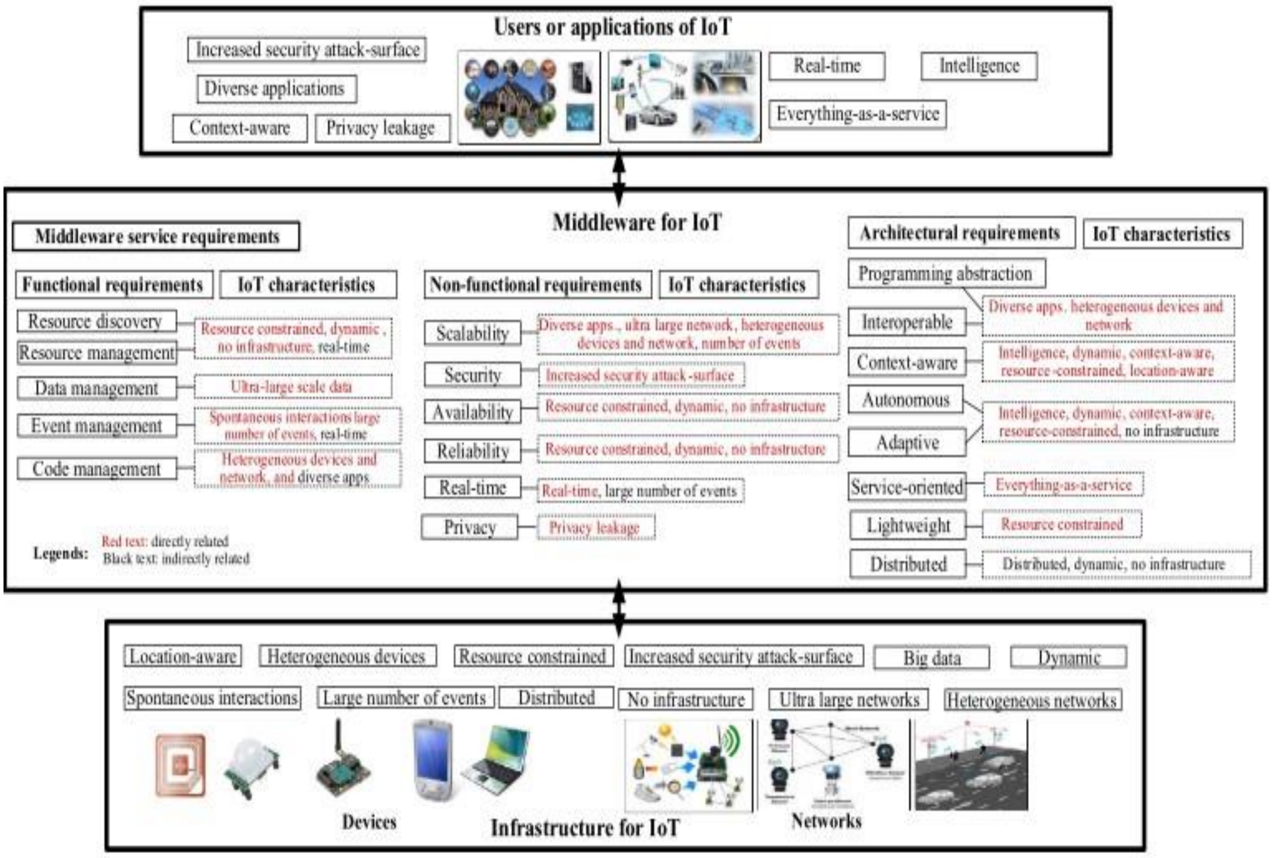


Figure 2.21: Classification of IoT devices centered on computational abilities, cost, and memory adapted from (Perera *et al.*, 2014)

Different middleware solutions have emerged to tackle the challenges of IoT. Li et al., (2015), classified middleware into three categories based on their design. This includes event-based, cloud-oriented, and Service-oriented. A consensus amongst research is that ideal IoT middleware should adopt the serviceoriented architecture that allows developers to integrate and deploy various bundles of IoT devices as services (A. H. H. Ngu M. Gutierrez and Sheng, 2016). Christen et al., (2014), identify middleware within the IoT vision as ideal to achieve scalability and supports the high level of interoperability through heterogeneity abstraction. According to Li et al., (2015), the application of middleware technology can satisfy the necessities for flexibility and reliability in data transmission. Lin and N. Bergmann, (2016) argue that all layers (from lower-level hardware to higher interface) of middleware should implement security and privacy protection. According to Razzaque et al., (2016) IoT middleware must provide device discovery and management, cloud connectivity (analytics and storage), Ease-of-deployment, Programming abstraction, and Popularity. In Sethi and Sarangi, (2017) an Ideal middleware offers an API for computation tasks, data communication, management, and security and privacy. Figure 2.22 summarises the middleware role for successful IoT platform implementations. It shows the interaction amongst the different middleware requirements, its device layer, and different features and services at the application layer.

Though many frameworks have large application instances, only a couple are open source, service-based, device and communication protocol agnostic support low-end MCU and being popular in smart home application with some reported success. As the smart-grid initiative is evolving in a developing context, a popular middleware must offer the benefit of extension and support as new smart grid cases develop (Wang, et al., 2014). In this regard, a couple of platforms within the competitive market stands out IoTivity released by the Open Connectivity Foundation (OCF), AllJoyn developed by AllSeen Alliance.



Users or applications of IoT
Increased security attack-surface
Diverse applications
Context-aware
Privacy leakage
Real-time
Intelligence
Everything-as-a-service
Middleware for IoT
Middleware service requirements
Functional requirements
IoT characteristics
Resource discovery
Resource management
Resource constrained, dynamic, no infrastructure, real-time
Data management
Ultra-large scale data
Event management
Spontaneous interactions large number of events, real-time
Code management
Heterogeneous devices and network, and diverse apps
Non-functional requirements
IoT characteristics
Scalability
Diverse apps., ultra large network, heterogeneous devices and network, number of events
Security
Increased security attack-surface
Availability
Resource constrained, dynamic, no infrastructure
Reliability
Resource constrained, dynamic, no infrastructure
Real-time
Real-time, large number of events
Privacy
Privacy leakage
Architectural requirements
IoT characteristics
Programming abstraction
Diverse apps, heterogeneous devices and network
Interoperable
Context-aware
Intelligence, dynamic, context-aware, resource-constrained, location-aware
Autonomous
Adaptive
Intelligence, dynamic, context-aware, resource-constrained, no infrastructure
Service-oriented
Everything-as-a-service
Lightweight
Resource constrained
Distributed
Distributed, dynamic, no infrastructure
Legends:
Red text: directly related
Black text: indirectly related
Location-aware
Heterogeneous devices
Resource constrained
Increased security attack-surface
Big data
Dynamic
Spontaneous interactions
Large number of events
Distributed
No infrastructure
Ultra large networks
Heterogeneous networks
Devices
Infrastructure for IoT
Networks

Figure 2.22: Middleware requirements enabling interaction from IoT device to application layer adapted from **(Razzaque *et al.*, 2016)**

2.4.7.1. Alljoyn

AllJoyn is a collaborative framework managed by AllSeen simplifying devices and application discovery and secured D2D communication (Jerabandi and M Kodabag, 2017). The AllSeen Alliance manages the AllJoyn open-source project with software code using open standards to enable the interoperability of IoT devices (Li et al., 2015). This middleware focuses on interoperability amongst IoT devices regardless of the transport layer, manufacturer enabling developers to write local/offline applications. Alljoyn embraces the servicesoriented architecture and provides features such as Onboarding Service to bring a new device onto WIFI network, Configuration Service, Notification Service (Jerabandi and M Kodabag, 2017). An AllJoyn network involves AllJoyn Apps and AllJoyn Routers, where every App is connected to a single Router. Figure 2.23 depicts a typical network illustrating the two versions of the AllJoyn framework. The AllJoyn Standard Core Library (AJSCL) and AllJoyn Thin Core Library (AJTCL), intended for resource-constrained embedded devices (Larsson and Nimmermark, 2016). The main drawback of the AJTCL is that it does not include a router daemon, so devices running the thin client must utilize the router of an adjacent AJSCL device.

AJTCL devices, the hosting bus segment, and the routers that handled all communication between all apps from a virtual bus are connecting using TCP (Wang, et al., 2014). The framework is developed to run on various platforms from Linux, Linux-based Android, iOS, Windows, and other lightweight real-time OS (Jerabandi and M Kodabag, 2017). AllJoyn presently provisioned ethernet, serial, PLC, and Wi-Fi. However, power-constrained wireless technologies, that is Bluetooth Low Energy (BLE) or Zigbee are not yet supported (Larsson and Nimmermark, 2016).

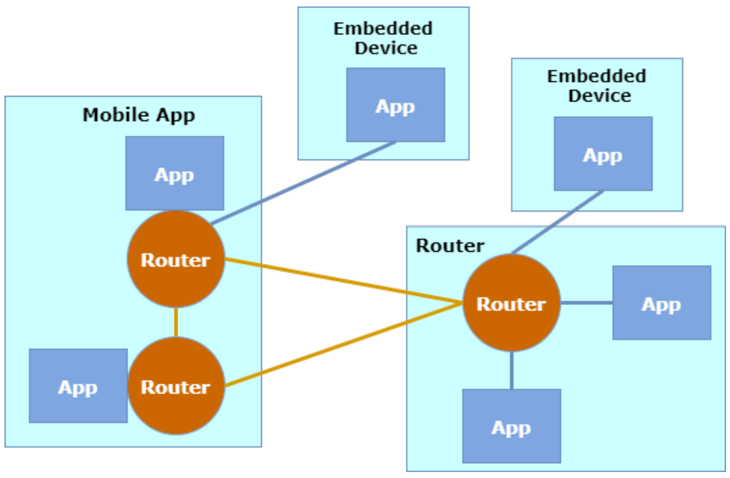


Figure 2.23: AllJoyn Network adopted from (Larsson and Nimmermark, 2016)

2.4.7.2. IoTivity

IoTivity is an IoT communications framework that enables smooth peer-to-peer connectivity amongst devices irrespective of the underlying OS or protocol satisfying several requisites of the internet of things for framework. It enables seamless device-to-device connectivity irrespective of OS or communication protocol to satisfy various requirements of IoT(Le, 2017). The Open Connectivity Foundation (OCF) develops specification criteria, interoperability guidelines, and a certification program for these devices(Kang and Choo, 2017).

Thus, IoTivity is a free reference implementation of the OCF specification maintained by the Linux Foundation(Larsson and Nimmermark, 2016). Although OCF intended to provision several vertical domains, so far it is primarily focusing on smart home (Lee, Jeon and Kim, 2016).

Several communication technologies (WIFI, Ethernet, BLE, and NFC) are supported by the Iotivity framework. The framework is available in many languages and supports various hardware platforms and OS; that is, several flavours of Linux, embedded Linux such as Tizen, Android, and Arduino(Lee, Jeon and Kim, 2016). As depicted in Figure 2.4.7.2-1, the IoTivity reference model provisioned both Rich and Lite devices. The service components are not supported, and the API only supports C base development. The instances of the lite devices are Arduino or ESP32 board(Jerabandi and M Kodabag, 2017). Both devices use CBOR to format and provisioned server resources for transport(Larsson and Nimmermark, 2016).

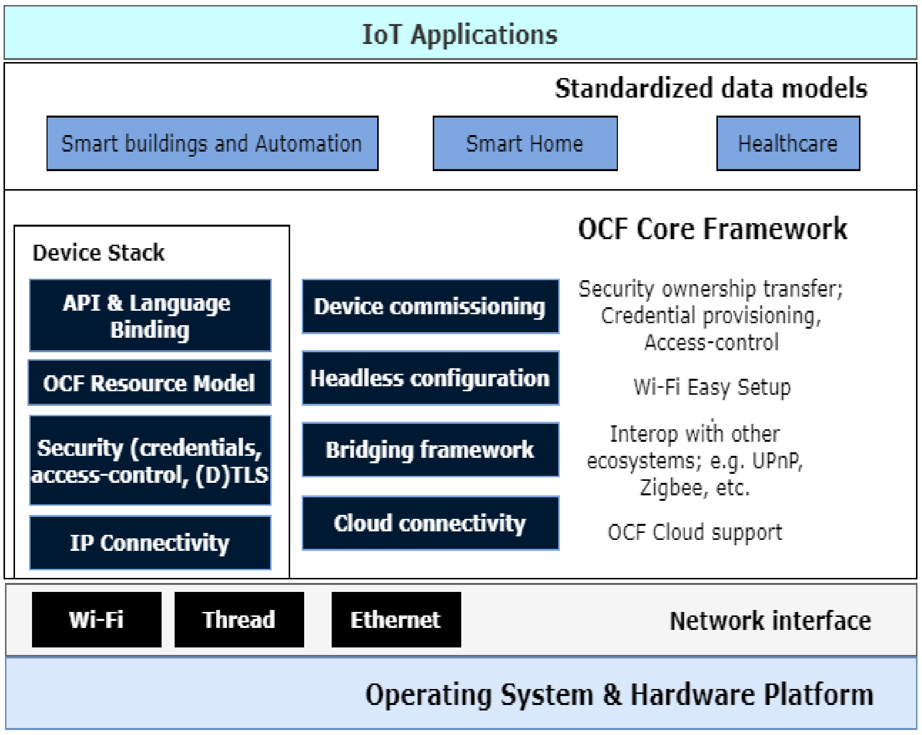


Figure 2.24: IoTivity Stack and architectures adapted from (Macieira, 2016)

The messaging model of IoTivity is structured around a resource-centered RESTful architecture in which everything (sensor, or devices.) is presented as resources using CRUDN (Create, Read, Update, Delete, Notify) model to handle resources by using CoAP protocol (figure 2.25) (Lee, Jeon and Kim, 2016). Alshadi & Kinder (2017) and Coval et al., (2017) list the feature below as essential building blocks of IoTivity:

- Discovery: Collective technique for device discovery (IETF CoRE)
- Messaging: Support for resource-limited device by default (IETF CoAP) also protocol translation using go-betweens
- Common Resource Model: Real-world objects defined as resources
- CRUDN: Request/Response mechanism using Create, Retrieve, Update, Delete and Notify commands
- Device Management: Network communication settings and remote feedback, reset, restart functions.
- ID & Addressing: For OCF entities (Clients, Servers, Devices, Resources)
- Security: Basic security for the network, access control based on resources, key management, etc.

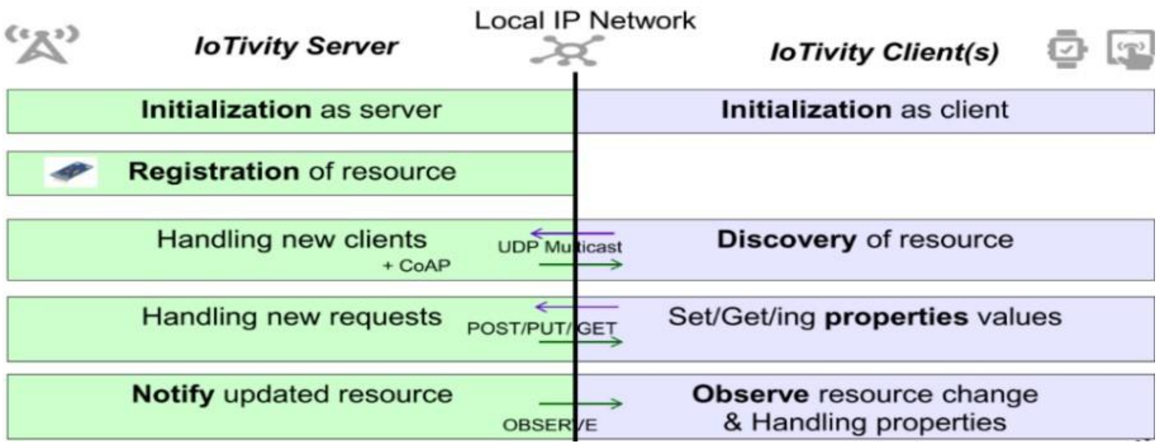


Figure 2.25: Client/server CRUDN interactions: Create, Read, Update, Delete, Notify Adopted from (Sun, 2017)

Being a CoAP oriented framework, most control in IoTivity is done mainly on the local area network (although cloud extension can be used) via UDP base requests. In figure 2.26 below, client UDP multicast requests are routed through IP 224.0.1.187 on ports 5683 and 5684.

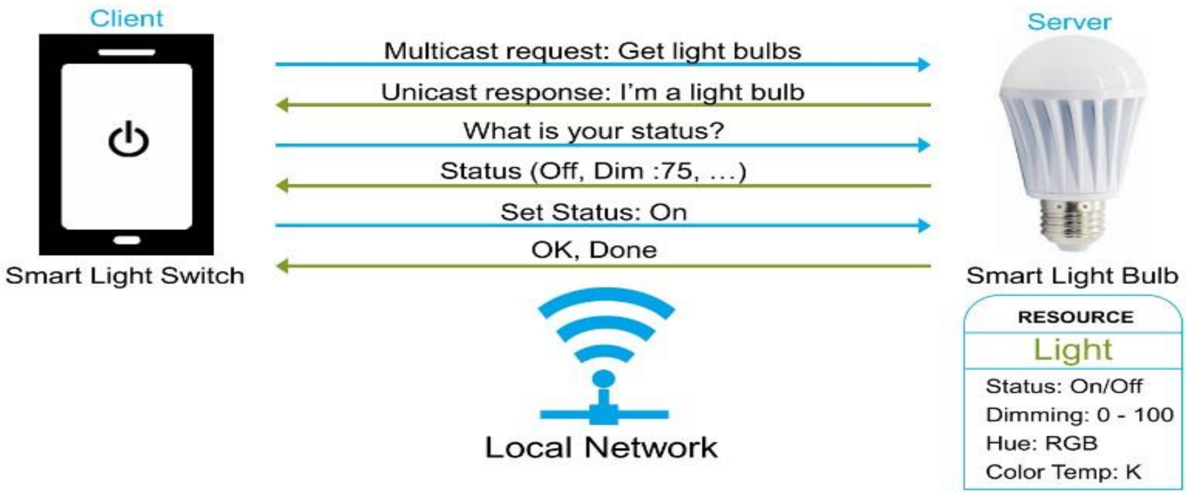


Figure 2.26: IoTivity client/server interaction adapted from (Maloor *et al.*, 2015)

However, IoT home-related applications require remote access to resources and devices. IoTivity has lately released cloud services within its stack to extend the accessibility of devices with authentication(Dang et al., 2017). Cloud services accessibility requires protocol conversion. IoTivity provides this via CoAP-HTTP proxy using the Java API or the more developer-centric NodeJS API (Coval and Sun, 2017). The later one, a JavaScript API for OIC (IoTivitynode) targets web development via a common RESTful architecture on the CoAP-HTTP proxy, therefore facilitating cloud development (figure 2.27).

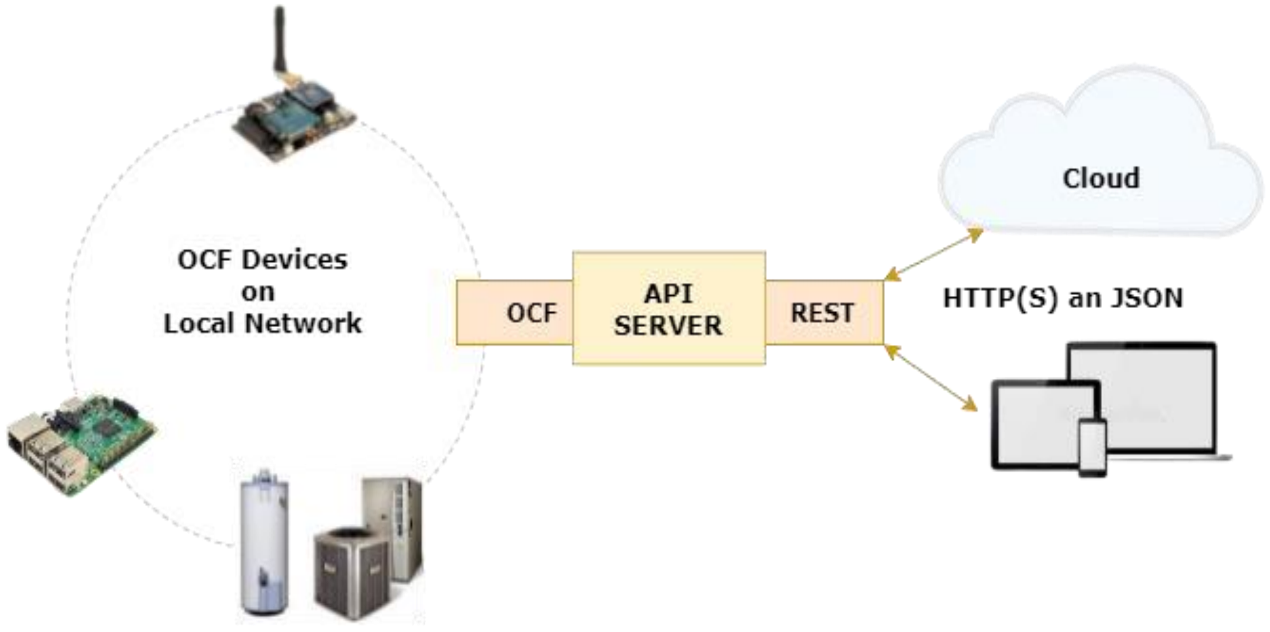


Figure 2.27: IoTivity-node concept (Macieira, 2016)

The IoTivity mainline stack is not compatible with lower category devices. It generally induced a higher processing and memory requirement. However, lower categories of device constraints have been vaguely defined because of the overwhelming variety of constrained devices that could be connected to the internet. To simplify these classifications, IETF has published an RFC 722 (Table 2.6) that uses three categories for resources restricted devices as shown(Maloor, 2017)

Table 2.6: Classification of constrained devices. adapted from (Maloor, 2017)

Name	*Data size (eg., RAM)*	*Code size (e.g., Flash)*
~~*Class 0, C0*~~	~~<< 10 Ki~~B	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

To cope with constrained devices, the OCF group released IoTivity-Lite (figure 2.28). IoTivity-Lite is a small footprint implementation of the OCF specifications (C and Java API) suitable for all device classes (~10 KiB RAM) over many OS including Windows, Android, and some popular RTOSes (RIOT, ContikiOS, FreeRTOS) (Maloor, 2017). Like the mainline implementation (IoTivity) IoTivity-Lite provides a RESTful design, messaging using CRUDN operations over CoAP, CBOR for data formatting, and DTLSbased security feature leveraging the mbedTLS library(Maloor, 2019). The transport layer uses UDP over IP (WIFI and Ethernet) network with development underway to extend it to Bluetooth.

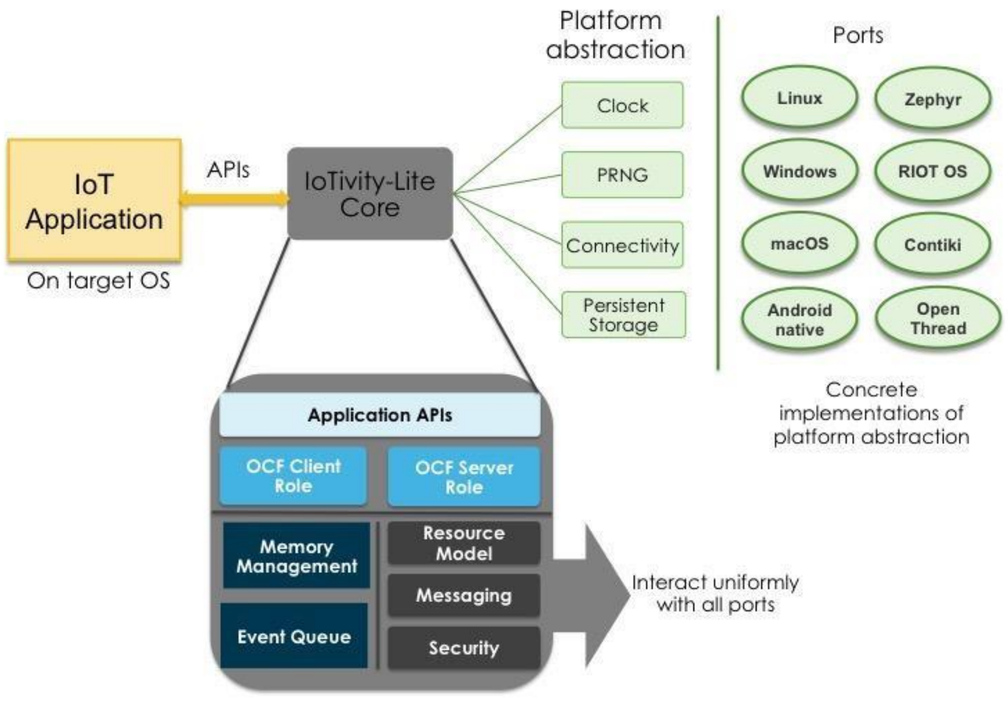


Figure 2.28: IoTivity-Lite framework adapted from (Maloor, 2019)

2.4.8. Cloud technologies

The Internet world is built around computers, centralized software infrastructures, or in the cloud (Wang et al., 2016). As stated by Asare-Bediako., (2019), the IoT vision is facilitated amongst other technologies by cloud computing. The cloud terms networked computers that allocate computing power, applications, and services to any computer or device on request (Da Xu et al., 2014; Lee and Lee, 2015). In recent times the interaction between IoT devices and the surrounding environment has expanded thus generating a huge amount of data to be handled (Faruque and Vatanparvar, 2015). Moreover, the resource-limited nature of IoT demands expensive hardware and software to store the bulk amount of data (Lin and Bergmann, 2016). To avoid these limitations, data communications in IoT leverage cloud-based computing infrastructures to accommodate Big Data requirements, provide protocols translation, data abstraction, and processing (see Figure) (Beligianni, Alamaniotis, and Fevgas, 2016). Initially, Cloud was used for the storage functions (Sahadevan et al., 2017). Nowadays, cloud computing provides development infrastructure, platform, software, and sensor network, as services (Faruque and Vatanparvar, 2015). According to Hu et al., (2017), cloud services are divided into three primary categories:

- Infrastructure-as-a-Service: IaaS offers extendable infrastructure, web devices, and storage spaces to users as services on demand. Cloud access is delivered through web service, API, command-line interfaces (CLI), or graphical user interfaces (GUI) (Naveen. Ing. Danquah. Sidh. & Abu-Siada., 2013).
- Platform-as-a-Service: PaaS offers to its users and customers a platform where to create and run their applications. Using PaaS platforms, developers can build and offer Web applications without directly handling the software requirements (download, installation, and configuration) needed. It ensures execution of users' given task at runtime(Jian *et al.*, 2017)
- Software-as-a-Service: SaaS delivers several kinds of applications plus interfaces for the end-users. A Subscription service usage offered by a free software dealer is delivered over the network in SaaS. The main advantage of this architecture is that an App can be developed and deployed without the need to extend the enterprise data(Dlodlo *et al.*, 2015).

Lately, numerous research ventures have focused on combining Cloud Computing and IoT to provide users improved services available anywhere at the same time ensuring scalability and security(Vinh et al., 2015). As stated in Risteska Stojkoska and Trivodaliev. (2017), the cloud "promises high reliability, scalability, and autonomy" for future IoT applications. That is, Cloud-based platforms support connectivity to the things making anything accessible in a time and space agnostic manner favouring user-friendless using the customized front end to access IoT and address the Big Data problems(Khatu et al., 2015). Furthermore, the cloud is an enabler for ideal IoT middleware. it can be used to extend and provide flexibility to IoT middleware deployment and to enable users to get improved understandings from the data collected by sensing devices(Sethi and Sarangi, 2017).

Numerous IoT cloud suppliers are presently emerging into the market leveraging appropriate and explicit IoT-based services(Ray, 2017). AWS, a reliable and affordable service, is one such cloud platform. It offers an efficient data storage mechanism, access to online servers anyplace worldwide leveraging any other application services accessible on the Internet (Sahadevan et al., 2017). Other studies have leveraged Google Cloud infrastructure offering cloud service to IoT applications (Abdulrahman et al., 2016). According to Naveen et al., (2013), the Google cloud platform allows designers to build and deploy Apps on Google's cloud infrastructure. It delivers disseminated storage and computing services. Google app engine is a PaaS providing software developers with a software developer kit (SDK) to develop web applications freely. Furthermore, it enables bi-directional massaging service between application servers and Android devices using the google cloud messaging (GCM)(Abdulrahman et al., 2016). The authors also discussed, google cloud datastore, a data storage service with a spread-out architecture that varies from the standard relational database regarding the manner it holds the non-relational data.

2.4.8.1. The case for the smart grid applications

As depicted in Figure 2.29, the challenges, and prospects of developing the next generation smart grids can be addressed partly by Cloud computing. For instance, cloud architecture can be used in a smart grid to handle the bulk of data processing (Big data problem) (Reka and Ramesh, 2016). Smart grid solutions can beneficiate to a greater extent by incorporating technologies such as cloud computing (Jose et al., 2016). Regarding smart homes, successful IoT energy management and HA applications aside from using middleware and gateways in local networks will need to incorporate the cloud as a unifying framework (Lin and Bergmann, 2016). An energy management cloud can gather consumption usage and then supports remote control and schedule the status of home appliances (Lee and Lai, 2016). As stated by Wang et al., (2015), the assimilation of WSN and the Cloud brings better flexibility, limitless resources, huge computing power, and the capacity to quickly respond to the user. Indeed, the cloud is proficient in monitoring, collecting, storing, abstracting, and processing data from devices maintaining secured standards. By analyzing this data, the cloud can prompt actions according to user-defined rules to achieve complex Smart Home control (Lin & Bergmann, 2016). According to Kedar and Somani. (2015), Cloud-based HEMS provide a daily demand forecast for the homeowners. Rathi et al., (2014), argued that process history such as real-time energy consumption, load management, time-of-use, amongst others, can be ported to a cloud database to manage the load and significantly contribute to the energy balance in the residential premises (Rathi et al., 2014). That is, a cloud platform effectively helps to analyze the state of several settings and controls in the home controlling their state anytime anywhere (Dey, Roy, and Das, 2016). As far as consumer behavior is concerned, cloud-based services allow for smarter decision-making. These complex decisions can be provided to things (or humans) allowing them to act and affect the environment (Coetzee and Eksteen, 2012).

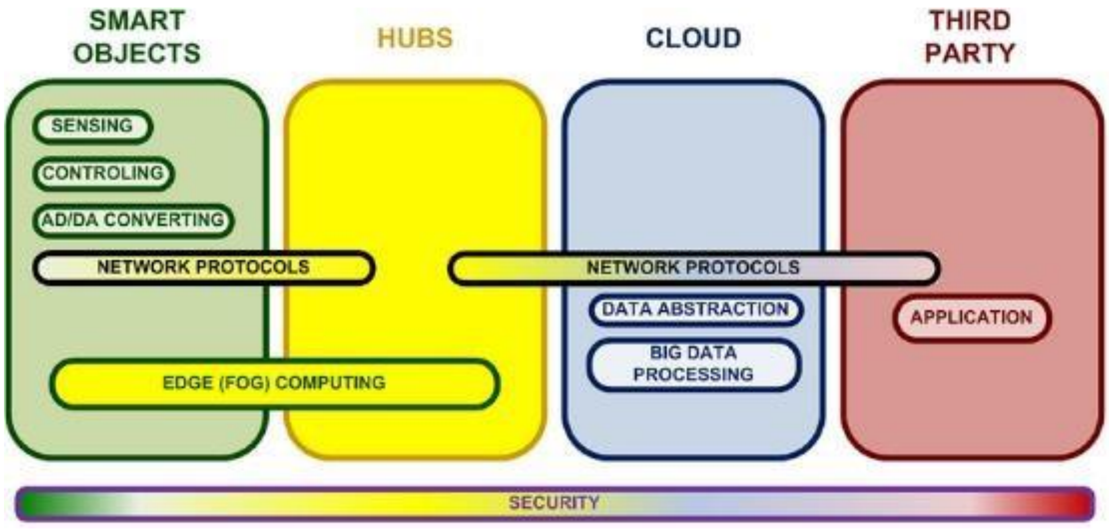


Figure 2.29: Typical IoT Smart home management model adapted from (Risteska Stojkoska and Trivodaliev, 2017b)

2.4.9. Mobile Technologies

The emerging 3G/4G/5G mobile communication technology and the Internet of Things yield the potential to effectively influence human existence (Liu et al., 2017). In this regard, the smartphone is a very handy and user-friendly device with a host of built-in communication and data processing features. Recently, the increasing popularity of smartphones among people has nurtured an increasing interest from researchers building smart IoT solutions (Sethi and Sarangi, 2017). Indeed, leveraging data from private or public cloud services, third-party services can be delivered to users via mobile App on smartphones (Fan et al.,2010). Moreover, mobile devices can be combined with Cloud solutions to offer users enhanced services that are accessible anywhere while guaranteeing scalability and security (Vinh et al., 2015). As stated by Christen et al (2014), Smartphone and mobile technologies are ubiquitous and easily operable and have the potential to extend IoT in different application domains and contexts. Dlodlo et al (2015), supported this view stating that in developing countries such as SA the high cell phone coverage offers the opportunity to bring services to remote locations. A smartphone can play a significant role in smart grid effort in managing residential load. According to Dey et al., (2016), a smart home is an automated home that relies on home automation leveraging amongst other mobile devices to control basic home functions and features automatically through the internet.

From the end user’s point of view, Internet-based HA is very convenient, flexible, and cheap (Pawar et al., 2016). In this regard, IoT-enabled web applications/mobile App on smartphones offer HA front ends to any users from any remote location (Korkmaz et al., 2015). Li et al (2015), summarised this stating that the combination of smart home systems and mobile devices is designed to help people take advantage of smartphones, tablet computers, and other mobile devices without restrictions of time and space to operate the equipment home. Mobile technology(smartphones) enhances IoT energy management activities with data anytime and anywhere (Thiyagarajan and Raveendra, 2015). Indeed, remote operation and data access are bringing the granularity of IoT context information closer to end-user attention, therefore, adding a new dimension to activity such as home energy management. According to Viswanath et al., (2016), a personal smartphone running cloud services can be leveraged in-home place to control and monitor appliance, receive and Utility DRM incentives (e.g., dynamic pricing), and for the user to send control commands (e.g., switch on/off plug) dedicated gateway. In HEMS, this is typically done via an Android mobile app augmented with a cloud messaging facility via push notifications.

Smartphones offer an extra level of sophistication in visualization tools. These displays not only show the overall home raw and tabular consumption but a disaggregated appliance level feedback (Liu et al., 2014). This type of feedback is mainly useful as it uses intuitive visual user interfaces (UI) to give consumers a deeper insight into their load. It enables users to learn more about their appliances thus, enabling their instinctive control based on their consumption profile (Risteska Stojkoska and Trivodaliev, 2017a). Via cloud technologies, the smartphone can enhance the bidirectional relationship between humans and objects providing a direct and effective platform for “human-in-the-loop” HEM applications (Stankovic, 2014). As stated in Li et al., (2015), the consumer can fully master the real-time and historical information about their consumption by accessing the IoT energy management platform through the web, Android (Li et al., 2015). Consequently, such interaction will provide feedback that helps engage consumer behaviour effectively.

2.5. Chapter summary

In this chapter was discussed the status of the traditional grid, highlighting the different challenges that these are facing in handling the increasing energy demand within the existing grids, particularly from the residential sector. Then, the literature on the smart grid vision focusing on IoT as an enabler for smart grid penetration in residential load management was covered. A review was performed regarding HEMS backbone (HAN) focusing on their architecture software stacks, and devices management. In this regard, the current state of the art in embedded design for IoT applications and the different implementations of IoT middleware as possible solutions for IoT semantic gaps are reviewed. Finally, a review of cloud computing and related technologies as enablers for IoT-related interventions for energy management was carried out.

CHAPTER THREE

Related Works and Solution Specifications

3.1. Introduction

This chapter builds on the literature review in the previous section to identify, discuss and compare related works around IoT-enabled smart grid interventions in the residential sector. In this regard, the research reviewed and compared works focusing on the implementation of IoT-enabling technologies for smart grid interventions in homes. For this purpose, the focus is on the implementations of sensor and actuator networks (within HAN) managed via middleware, augmented with BaaS cloud technologies extended over smartphone technologies to develop and implement an interoperable, scalable, affordable, and performant platform for smart grid DRM interventions in homes. Therefore, in this section, the aim is to review and compare similar works so to identify and define the specifications of the technological tools that will enable us to develop and implement a platform suitable to this study context.

3.2. Related Works

IoT is a novel ICT paradigm showing interest from various studies regarding IoT platforms for HEM. *Saga K N & Kusuma, (2019),* designed and implemented a home automation system that leverages IoT to control most household appliances over an easily adaptable web interface. The planned system offers great flexibility by using Wi-Fi technology to connect its spread-out sensing devices to a home automation server. Such an implementation aimed at decreasing the system deployment cost and facilitate future upgrades, and reconfiguration.

Also, using a web browser interface from any local PC within the HAN via the server IP or remotely using a PC or a mobile device connected to the internet one can access the automation system. The authors argued that WIFI was selected to increase the system security (via a secured WIFI connection) and to improve system flexibility and scalability. Nevertheless, this works does not handle the communication between devices on the local network, a single server gateway to which home appliances sensors and actuators are connected. This setup is archaic and incurs a scalability issue. Furthermore, the need to connect to the home gateway via its IP requires private DNS which is restrictive in many contexts as this suggests a payable subscription to some ISP. Here the cloud is used as SaaS to forward an email notification to users. However, the non-real-time nature and textual format of email limit the depth of feedback and analytics that can be done on consumption data. Moreover, this work lacks a middleware to manage devices on the network, provide interoperability and improve scalability which paramount factor in any effective IoT-based HEM platform. Another noticeable issue here is that the nonstandardized architecture will increase security concerns and increase deployment and maintenance costs.

Kim et al., (2015), proposed an IoT-based DHAN for HEMS platform around ad hoc P2P networks between the home appliance and Nomadic Agents (NA’s) running on the user smartphone. The authors implemented a middleware (OSGi) to manage and provide service within the HAN energy saving (sensing and actuating) device (ESD) and the user smartphone working as a client or dynamic gateway over Bluetooth. In their work, the authors proposed a platform that aims to mitigate the static architecture of ZigBee based HAN composed of several fixed GWs gateway ‘always on’ using ad hoc Bluetooth based networks connecting a nomadic GW to a home appliance to transmit consumption data to a central management server (CMS) that aggregate the consumption of several households and provide analytics on user consumption. However, the work presented here add a level of dependency to the HAN as the smartphone become a network resource. this implies that the cell phone because a central device needed to be on during critical communication session. In this regard, smartphone power consumption becomes an important factor as wireless communication and the different energy services (provided via the nomadic agent) add additional load to an already application-intensive device. This may incur discomfort to the user, which must be notified and constrained to maintain an acceptable battery level. Although the use of smartphones is motivated by their ability to support emerging wireless technologies, the range limitations and signal interference of Bluetooth technologies in domestic places still required some level of signal amplification and relays for effective communication and to avoid data losses. Moreover, the lack of an emerging middleware but instead an architectural dependant does not present any form of protocol integration for interoperability which is a requirement for IoT platform for HEMS. Furthermore, this works lacks the cloud interface that provides energy management applications at anytime, anywhere monitoring and appliance control constituting a powerful motivator for HEMS efficacity and acceptance and cost of these systems.

In *Beligianni, Alamaniotis, & Fevgas, (2016)*, a software architecture for efficient and secure energy management within the smart grid that leverages the recent developments in Smart Grid, IoT and Fog and cloud computing to deliver energy services load and price forecasting as well as handling the big data challenges of IoT and latencies and cost of cloud infrastructure is proposed. At the heart of their platform is the IoT gateway, design over a low-cost embedded device(raspberry-PI) running the Eclipse Kura framework, a scalable free IoT Edge framework built on Java/OSGi used as middleware to offer hardware abstracting (protocol interoperability) via an API that gives access to the hardware interfaces of IoT Gateways. Moreover, it connects the gateway to cloud infrastructure (edge and cloud platform) via the Mosquito MQTT. This enables the platform to push the stream of smart meter data to a message broker at the edge for analytics and knowledge extraction and further, push the aggregated data securely to the cloud preserving privacy. In accordance with the current trend in an ideal IoT platform for energy management, this work, make used of Gateway, cloud (over fog computing) infrastructure to provide energy service to the consumer. However, less attention is given to the management of HAN network devices as related to devising management, discovery, and security. Furthermore, the middleware being used required a higher category (see figure 2.3.6.5-1) embedded systems that can run the Java Virtual machine (JVM). This has the drawback of mitigating the expected miniaturization of IoT implementation, increase cost, and limit scalability.

Korkmaz et al., (2015), proposed a HA system leveraging emerging open technologies providing a platform for multi-home automation via enabling opensource cloud infrastructures and web applications running on user’s Android smartphones as well as via a website. The focus of the authors was to offer cost-effective home automation as a service via open-source cloud services (GCM) and seamless integration to consumer life mainly using ubiquitous mobile technologies. Although this work integrated IoT enabling technologies (cloud services, smartphone for feedback, and remote management), the system architecture used here is not participating in the standardization effort for ideal IoT implementation. Moreover, the lack of middleware to manage local devices on the HAN further heterogeneity and scalability within this implementation. This is further highlighted in that the local hardware within the HAN, mainly consists of the i.MX53 card, a 1 GHz ARM cortex-A8 processor, and its 1GB DDR3 main memory connect to the appliances either via its wired or wireless interfaces. As shown in figure 2.3.6.5-1, this hardware sits within the higher category of IoT devices, thus limiting miniaturization and increasing deployment costs.

Lee and Lai, (2016), propose an energy management cloud platform to provide energy management based on the Software-as-a-Service (SaaS) cloud model. To enhance interoperability, the authors propose a universal smart energy management gateway based on a free Internet of Things (IoT) framework named IoTivity to monitor and manage IoTivity-compatible devices. In this work the author used the IoTivity middleware to abstract from the monolithic, ad hoc implementation that locks traditional HEMS to private protocol or mechanism limited the choice and spectrum of possible devices to implement IoT HAN. Therefore, the main goal of the author here was to tackle of ever so common issue of interoperability and device management within IoT based HAN network provided a completed architecture that handles the platform requirement for data communication and management from appliances on the HAN to services provided in the cloud for local or remote management of consumer load. However, because IoTivity is CoAP based framework, the Authors proposed a REST framework for bridging CoAP to HTPP to access their dedicated cloud infrastructure. For further scalability and web interoperability as required for ideal IoT, energy management platforms using popular open-source REST API and cloud services facilitate grid service interfacing, reduce the cost of technologies, therefore increasing the penetration of such platforms in homes.

Al Faruque & Vatanparvar, (2016), proposed a fog computing-based platform for energy management focusing on interoperability, scalability, adaptability, and local and remote monitoring while leveraging open-source software/hardware featured to allow users to implement the energy management with the customized control-as-services. The authors focused on facilitating the deployment of their platform in residential places by mitigated the cost associated with computing devices, software stack, and communication devices. Thus, they focus on using popular, open-source hardware within their HAN, in this regard, The Raspberry PI acting as a home gateway and TelsoB mote running TinyOS and communicating over wireless Zigbee or Bluetooth or wired Ethernet and serial have been used as network devices. To support device-to-device communication, security, and device management within their heterogeneous platform, the Author used the Devices Profile for Web Services (DPWS) middleware centered on SOAP-over-UDP, SOAP, WSDL, and XML schema to abstract the management of HAN devices and provide web connectivity. Through this platform, the authors proposed HEM as a service (monitor, control) on Fog via the HEM control panel forwarding DR signal to a local home gateway and provided a web page as front-end to users. Though the Author advocated the use of a middleware within the local HAN, the use of the home gateway for protocol translation complicated network architecture, and the cost of hardware has such gateway are generally high-end devices requiring more processing power thus consuming more power themselves. Moreover, A Web interface is provided based on the local router DNS info. This limits operation on the local network or increases the cost of implementation when an ISP subscription is required for remote control operations. Furthermore, the cloud is missing as fog only processed that on the edge. Fog computing transfers the paradigm of cloud computing further to the edge of the network. It is a platform that may also offer IoT with the ability of data preprocessing while meeting real-time requirements. Fog needs to include cloud for increase servicing in IoT platforms and increase acceptance of IoT platform as remote, anytime anywhere monitoring control is made possible.

Sahana et al., (2016), proposed a sample implementation for home energy management and control providing consumers with a web-based interface to observe the power consumed by many appliances leveraging the internet to control home appliances from anywhere in the world. this is achieved by leveraging the Internet of Things protocol stack comprising emerging IETF open standards-based such as 6LoWPAN, RPL, and CoAP to allows unified integration of the appliances at home to the current internet infrastructure. In this work, the author develops a HAN based on low-cost, popular hardware and software technologies while proposing an architecture that offers interoperability and scalability. The author used 6LowPAN wireless technologies to handle communication within the HAN. To reduce the implementation cost of their HAN, an embedded device has been used. The TI CC2538, an ARM Cortex-M3-based MCU system with up to 32KB on-chip RAM and up to 512KB on-chip flash running the contikiOS communicated via 6LOWPAN have been used to sense and control home appliances. a BeagleBK, running Ubuntu 14.04 acts as a gateway connecting the 6LoWPAN network to an Ethernet network. Although this work incorporated popular and studied IoT technologies, it still lacks the enabling technologies to make it scale to different households and interface to grid services outdistancing the security issue related to such a monolithic implementation. According to the analysis of the existing smart home platform above, it transpired that the platform for these systems is ad hoc or close/monolithic. That is each system is on a single system without architecture that embedded recognizes IoT enabling technologies from WSN to cloud-based services accessible on user mobile devices.

To achieve the above demand, *Li, Nie, Chen, Zhan, & Xu, (2015)*, proposes a framework for energy management applications running on a home gateway and energy service systems for multi-homes running on Azure cloud leveraging dedicated 3rd party energy service providers. Each home is with a gateway (Intel(R) Core (TM) i3-2100 CPU) powered with Microsoft Lab of Things (HomeOS) middleware. Windows Azure cloud technology is used for data management to realize multi-family management. For front-end requirements, the Author provided an Android mobile terminal and Web using publish/subscribe MQTT model and azure push notification. The MQTT message middleware enables the realization of a reliable diffusion of data and command across the sensing layer, transmission layer, and application service layer in the third-party cloud. The authors aim to abstract away the limitation of monolithic IoT platforms by providing an architecture that incorporates within layers the different enabling technologies of IoT in the smart grid context. Using the cloud as both SaaS and PaaS the author proposed a platform that scales and fits a diversity of deployments. However, the use of the LoT middleware requiring the Homes to increase the processing power, power consumption, and cost for each home gateway. Being a gateway dedicated middleware, LoT (HomeOS) limits the miniaturization of devices in the HAN increase the cost of this implementation. The Use of MQTT instead of the more popular CoAP for communication on HAN is another limitation for interoperability and scalability.

Viswanath et al., (2016), propose an IoT platform targeting residential consumers leveraging smartphone and cloud technologies to offer Smart grid empower energy management (DRM signal) and home automation as services. In this regard, the authors proposed the IoT elements, protocols, and the testing setup for IoT context together with the software designs that have been used for consumers’ energy usage patterns feedback and control focusing on the response time and the ability of the platform to handle many users. To accomplish this, the authors propose, a UHG responsible for the transmission of collected data to the cloud via the network layer. This is accomplished using the popular Raspberry pi computer, an IP-based system and runs in HTTP and XMPP protocols, used as a translator to interact with other non-IP-based devices in the system. Openfire server as a middleware on the Gateway to provide uses pub-sub mechanism to push information to subscribers. To display real-time information (e.g., Dynamic Pricing) and for the user to send control command (e.g., switch on/off plug) to the UHG, native push to smartphone devices through GCM was implemented in the testbed and using RESTFUL HTTP periodic uploading of sensor node data is done from the gateway to the cloud platform. XMPP is TCP-oriented which expensive for lower-end device notwithstanding that it does not recommend with IETF standard for IoT. Moreover, XMPP is a heavyweight protocol streaming XML (with less interoperability than JSON) with a specification that has no complete implementation yet. Openfire essentially lacks functionalities such as discovery, provisioning, security which IoT middleware provides. Also, in this application security and permission is solely on the cloud this All of this is provided by middleware. Furthermore, Openfire is not supported on lower-end devices thus increase the cost.

3.3. Solution Specification

An energy management cloud platform is proposed to provide energy management based on the Software-as-a-Service (SaaS) cloud model for deploying energy services and BaaS for platform backend requirements. The proposed architecture is based on standard open-source protocols, services, and development tools. An overview of the proposed architecture is depicted in Figure 3.2. A three-layer platform consisting of a HAN gathering consumption data and controlling appliances, a home gateway (D2D and D2C connectivity), cloud computing, data storage was proposed. This architecture further offers services for both home gateway and consumer over the third party as well as a smartphone App as user front-end for enhanced feedback. Additionally, the cloud provides an interface to smart grid services as there are made available by smart grid third parties.

3.3.1. Software components

To guide the development of the testbed, a certain number of technologies facilitating IoT application development and deployment in smart homes will be adopted. Adopting state-of-the-art solutions, the wok targets open-source software technology to alleviate the complexity of proprietary software and the related cost.

3.3.1.1. Middleware for the platform HAN management

As to focus on research goals, the IoTivity platform will handle local networks' interoperability, scalability as well device management complexities using the IoTivity framework. Therefore, IoTivity will handle resource discovery, device management, protocol conversion, and security requirement for the platform. IoTivity-Lite the OCF release for the constrained devices was recently released primarily devices within category 3 (Figure 2.21). Therefore, an adaptation or port needed to be developed to support lower category devices.

3.3.1.1.1. IoTivity-Lite arduino port

To sustain the goal of low cost within the platform, IoTivity-Lite framework was ported to lower category (lower cost) devices (category 1&2). For this, the popular Arduino MCU and the Espressif ESP32 Wi-Fi MCU were targetted. However, the Port of the IoTivity framework, rely on OS running on the MCU. Based on the literature review on RTOS and the state-of-art FreeRTOS and ContikiOS were considered for being popular RTOS for low power, low-cost MCU. ContikiOS was used on the Arduino MCU because of its low memory footprint and simplicity in developing firmware that is seamless to IoTivity-Lite integration which uses ContikiOS itself within its stack. The ESP32 MCU, boasting a higher memory footprint of ~500Kbytes of RAM. In the case of the ESP32, an adaption of the Initial IoTivity port based on the FreeRTOS OS was used. Figure 3.1 below, shows the architecture for IoTivity-Lite Arduino port.

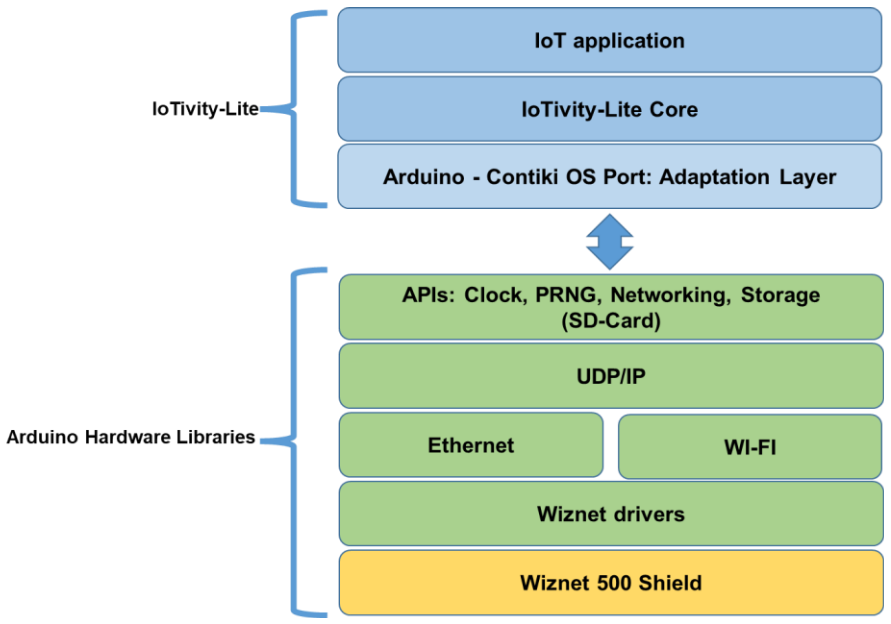

IoTivity-Lite
IoT application
IoTivity-Lite Core
Arduino - Contiki OS Port: Adaptation Layer
Arduino Hardware Libraries
APIs: Clock, PRNG, Networking, Storage (SD-Card)
UDP/IP
Ethernet
WI-FI
Wiznet drivers
Wiznet 500 Shield

Figure 3.1: Porting IoTivity-Lite to Arduino MCU adapted from (Maloor, 2019)

3.3.1.1.2. High-level devices management and cloud connectivity

The OCF IoTivity group avails a JavaScript port of the IoTivity stack running on the Node engine or IoTivity-node for a high-level device. Using the IoT-rest-APIserver, a NodeJS REST server for HTTP-based communication using IoTivitynode as a client, a device-to-cloud interface was established with the local CoAP devices offering those services or remote access on client App (mobile or desktop app). Connecting is provided using the IoTivity-node empowered IoT-rest-API-server on the gateway device (Raspberry PI).

3.3.1.1.3. Communication technologies for the proposed solution

In the HAN communication between IoTivity devices is around Wi-Fi primarily. However, Ethernet is used for demonstration purposes as the Wi-Fi shield for Arduino was not available. Wi-Fi and Ethernet are a communication technology that is ubiquitous to residential places. Thus, leveraging ICT infrastructure in a domestic environment, WSN can be simplified and made cost-effective. Wi-Fi decreases the deployment cost and will increase the ability of upgrading, and system reconfiguration, and high-end security mechanism. WIFI is selected as being advantageous due to its higher bandwidth, large coverage, easy expansion (M. Khan, Silva, & Han, 2016). Communication within HAN devices on the application layer is handled by the middleware using CoAP over UDP. RESTful HTTPS is used for cloud communication, publish, and subscribe via Parse server Live Query mechanism over Back4App BaaS tools.

3.3.1.2. Cloud deployment for the proposed solution

In this work, the cloud is mainly used as Software-as-a-Service (SaaS), for energy data storage and as an energy services provider for energy monitoring and the management and control of appliances on the platform. Based on the literature, design requirements, the open-source Parse Server was used. Parse was used as a server to provide a RESTful API for a plethora of devices on the different programming languages. Parse server is flexible and can be hosted and migrated from one cloud platform to another. Though Google Cloud and Amazon are the most popular in terms of cloud Hosting, there are not native Parse server environments for pub-sub mechanisms which central to IoT platforms for smart home applications. The back4App cloud platform was chosen to provide computing, storage (mango DB), server management, Live Query, cloud background Jobs and third-party login (i.e., Facebook), and mobile push notification (mainly Android) all as BaaS for an IoT platform centered around a mobile or web application.

3.3.1.3. Mobile development for proposed solution

Smartphones are central to the front-end requirement of an ideal IoT platform for the smart home. In the implementation, smartphone is used for energy monitoring (enhanced feedback) and HA anytime and anywhere as well as to display smart grid incentives when the case is made operational within the South African context. In this regard, an Energy App for Android devices is developed. the App is cloud-based using Parse server BaaS and provides both energy monitoring and HA using Live Query and Android Push notifications mechanism. Again, being a Backend server for mobile application parse was thus used as the interface between the front-end and home cloud services. Android development was performed on Android Studio using Parse Android API and the Java programming language.

3.4. Chapter Summary

In this chapter was covered the specifications that govern the architectural designs and experimental work for the platform. The research went through the rationale behind the selection of the different tools for the proposed solution as the review in chapter 2. Moreover, it covered the new solutions that were developed or adapted to satisfy the research objectives.

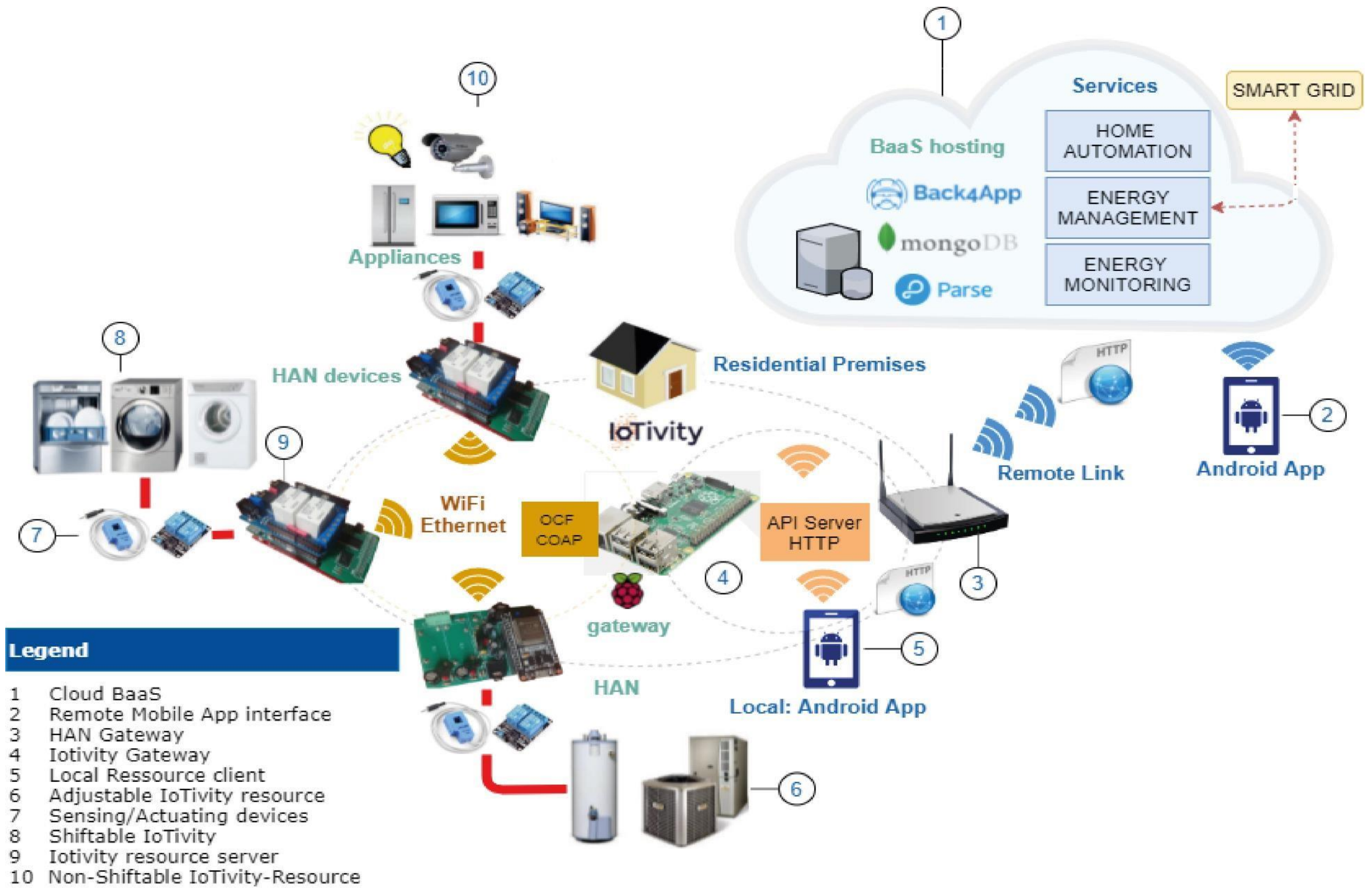


Figure 3.1: Proposed System architecture

CHAPTER FOUR

PLATFORM DESIGN AND DEPLOYMENT

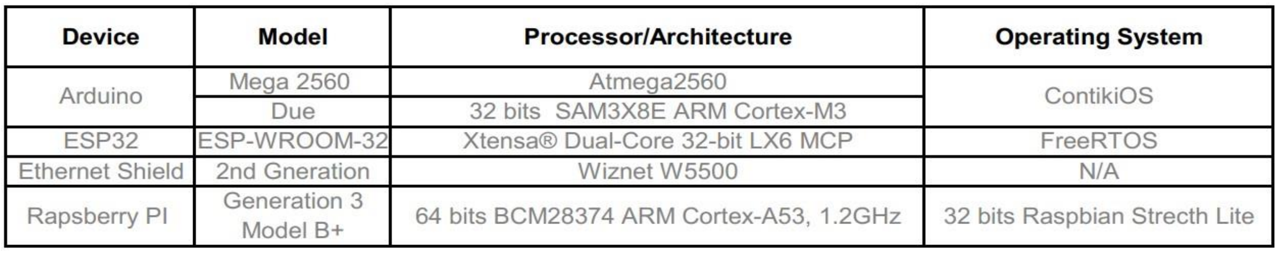
4.1. Introduction

This section describes the design and deployment of an experimental platform for energy management. Leveraging this platform, an energy application on an Android smartphone will be developed, tested for energy monitoring and HA. Integrating cloud computing, pub-sub mechanism, and cloud Job, and the HAN gateway in a home, a peak load management algorithm is implemented to manage consumption in the residential place. The architecture for the experimental platform followed by a rundown of its core components as well as the case study implementing the different objectives for the research are presented. The evaluation of the overall experimental platform according to research questions and objectives will be based on a methodology, relying on scenario testing and response time throughout the entire architectural layers. Secondly, are presented the results from the different experimentations and provided a throughout the discussion of observations made.

4.2. Equipment for an experimental platform

The experimental platform leveraged open-source embedded devices with wide support (development libraries) to facilitate an efficient design of the testbed. The hardware is comprised of server device interfacing sensing devices (CT and PT sensor) to provide consumption data in form of current and power consumption of different appliances or groups of appliances. The system will be composed of a single gateway device interfacing with the home router. This simplifies the security aspect of the system. All devices on the HAN, are selected based on their support for the IoTivity framework, flexibility in firmware design, availability, and affordability.

Table 4.1: **Device used for IoTivity network (HAN)**



Device	Model	Processor/Architecture	Operating System
Arduino	Mega 2560	Atmega2560	ContikiOS
	Due	32 bits SAM3X8E ARM Cortex-M3	
ESP32	ESP-WROOM-32	Xtensa® Dual-Core 32-bit LX6 MCP	FreeRTOS
Ethernet Shield	2nd Gneration	Wiznet W5500	N/A
Rapsberry PI	Generation 3 Model B+	64 bits BCM28374 ARM Cortex-A53, 1.2GHz	32 bits Raspbian Strecth Lite

As shown in Table 4.1 above, The HAN uses Arduino mega 2560 and Arduino

Due providing connectivity via Wi-Fi. However, for the reason of availability and affordability, Ethernet was used in this implementation over the Wiznet W5500 shield. Nevertheless, Wi-Fi was used to provide connectivity for the ESP32. To provide a sensing and actuating interface, design and manufacturing are carried out for plug-play shields comprising current and voltage transformers. When assembled to the Arduinos and ESP32 this constitutes the IoTivity-Lite powered local server handling resources such as power and current information from household appliances. IoT HAN around the Raspberry Pi takes advantage of a large developer community and open-source software. The Raspberry Pi is well supported by IoTivity for constrained and Rich devices making it ideal for interfacing constrained device networks with the internet. Moreover, its small size makes it non-invasive and cost-effective. The Raspberry Pi 3 as the platform HAN's gateway is used.

4.2.1. Platform motes shields

The HAN's devices are augmented with motes designed and manufactured (Altium designer) as plug-and-play shields. These shields provided the sensing and actuating interface to existing home appliances via noninvasive and safe electronics devices. The design was carried out as shown in figure 4.3 and figure 4.4. The completed manufactured and assembled motes are shown in figure 4.5.

4.2.1.1. Current and voltage sensors

An electric signal, either an analog voltage, electric current, or digitally encoded output is generated proportional to the current that flows through a conductive element for the current sensor. According to Blanco-Novoa et al., (2017), there are mostly four types of current sensors, that is, Ohm's law sensor, sensors based on Faraday's law sensors, magnetic field-based sensors, and Faradayeffect sensors. For the testbed, the magnetic field-based sensor in the Current Transformer (CT) as well as voltage transformer in the form of AC-AC transformer, depending on availability will be used measurement. In this regard, the current sensor from YHDC, a Chinese company heavily involved in electric power equipment is selected. The SCT-013-030, an inductive, ferrite split-core current transformer supporting a primary current of up to 30 amperes (RMS) with a proportional output voltage of 1V (RMS) is provided. The sensor used has an embedded burden resistor of 62 Ω outputting a voltage proportional to the current. The sensor output the voltage with an error tolerance of 1% of the measured value and is equipped with an audio jack plug for interfacing. According to Miron-Alexe, (2016), the SCT-013-030 is a robust and suitable sensor for industrial and household applications not requiring a high grade of accuracy.

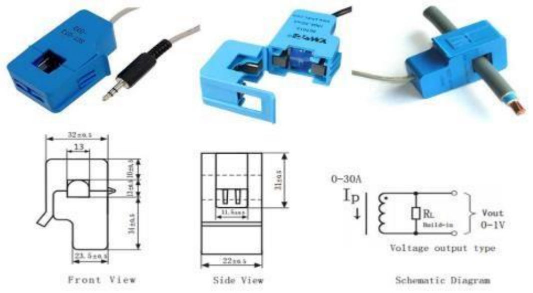


Figure 4.1: Current Transformer (CT) adapted from (Miron-Alexe, 2016)

Figure 4.1 represents an SCT-013 split-core transformers with a clamp-on mechanism, describing its physical characteristics and its electrical schematic. On the electric schematic, Ip is the current through the primary winding and RL is the embedded burden resistor whereas Vout is the output voltage of the CT (1Vrms). The voltage transformer (figure 4.2) from Mascot rated 230-240 V AC at 50 Hz outputting a 12 V/500 mA AC is used to measure the mains voltages for power calculation. According to Wall, (2016), this transformer exhibits a phase lead changing from 4° at the lower edge of the supply range up to 7½° at the upper limit.

Figure 4.2: Mascot ac-ac 230V/12V 500 mA voltage transformer adapted from (Wall, 2016)

Figure 4.3 depicts the schematic circuit for the sensing system. For voltage measurement, the 12Vac signal from the transformer is conditioned for the Arduino ADC and stepped down using the voltage divider (R7 and R8) which limits the voltage at A1 to 1.091Vac (RMS). The voltage follower based on the high impedance op-amps (LM358) is used to condition the before the ADC input. The LM358 input for voltage reading using a DC offset that lifts the signal reference to 2.5V via the voltage divider (R9 and R10). For voltage measurement, the LM358, for the high impedance it offers at the ADC input, thus increases stability and accuracy of the reading at A1. For current sensing, the LM358 simplifies interfacing with the MCU by eliminating the negative part of the signal outputting a half-wave rectified signal (CT2).

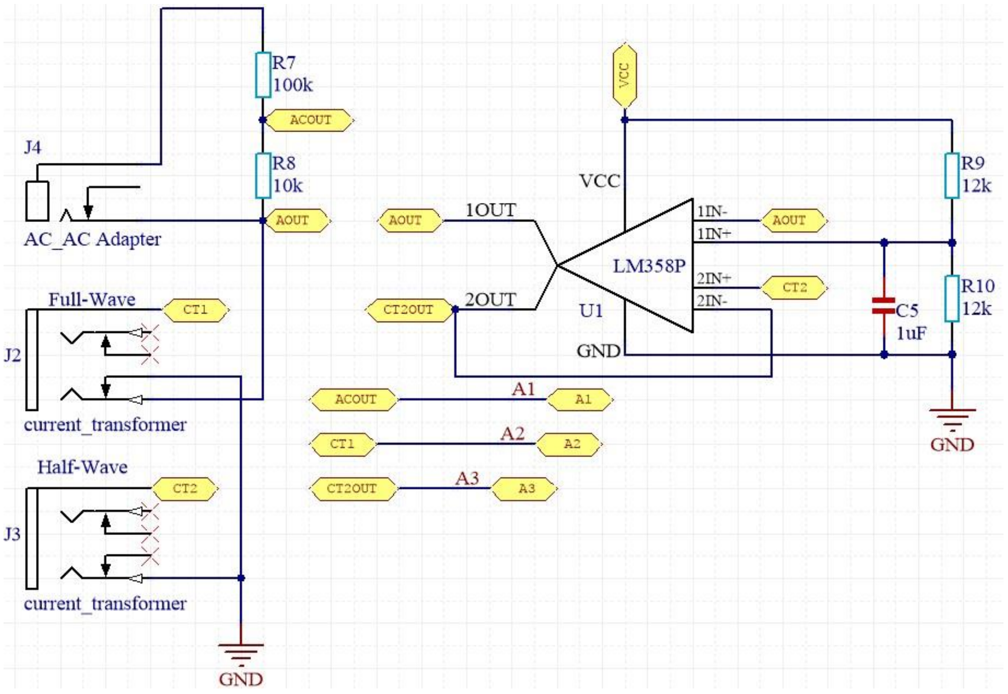


Figure 4.3: Schematic defining the wring of the SCT-013 current sensor Adapted from (Miron-Alexe, 2016) and (Sutisna et al., 2019)

The main advantage of this circuit is in the stability and accuracy of the current reading. However, it was noted that the prior is only true for symmetrical ac signal. Moreover, measuring low current is not ideal with this circuit. Since the op-amp gain is 1, low voltage (µV) is directly read by the ADC which zeroes such a low signal through the 10 bits ADC resolution on the Arduinos. To counter that, a 2.5V DC offset was added to the current signal to obtain a full wave rectified signal (CT1). The sampling graph for both the half-wave and fullwave current sensing can be seen in figure 4.6. The voltage and current reading from both the CT and AC-AC adapter is a sine wave whose root mean square (RMS) value can be obtain from equation (4.1) (Serov, Serov and Makarychev, 2019). Thus, the RMS is computed on each signal at instant t while sampling the signal during an interval T. In Equation (4.1), t0 is the time instant when sampling begins.

$$S_{rms} = \sqrt{\frac{1}{T}\int_{t_0}^{t_0+T} s^2(t)\, dt} \tag{4.1}$$

Where S_{rms} can be either RMS current I_{rms} or RMS voltageV_{rms}. Since ADC are used to convert analog signal into series of digital samples. The measurement based on averaging the squares of the ADC input signal samples is the mostly used. From this method, the RMS value is computed as in equation (4.2) after discretization of equation (4.1)(Albu and Heydt, 2003).

$$S_{rms} = \sqrt{\frac{1}{N}\sum_{n=0}^{N} S_n^2} \quad (4.2)$$

Where N represents the number of samples for the period T. based on work from (Learn | OpenEnergyMonitor, 2021) related to resistive load, using both instantaneous current and voltage, the real power and apparent power were calculated using equations (4.3) and (4.4).

$$P_{real} = \sqrt{\frac{1}{N}\sum_{n=0}^{N} I_n \times V_n} \quad (4.3)$$

Where N is the number of samples over period T and $I_n$and V_n the instantaneous current and voltage.

$$S = I_{rms} \times V_{rms} \quad (4.4)$$

4.2.1.2. Actuators

Relays provide actuation of appliances via the mote. According to, BlancoNovoa *et al.*, (2017), using a relay, appliances can safely be controlled via the electrical insulation that can be created between a low-voltage circuit and higher voltage circuit to which High current devices are connected. Among the different types of models, considering the current rating of the appliances in the case study the OMRON, G5LE-1-E 12VDC at 16A 250V AC relay is selected. This relay tolerates a maximum switching power of 4 kVA and has a low-coil power consumption(400mW) as well as a low price (around R35/units). Moreover, the small form factor of this relay makes it suitable for small form PCB design thus increasing the miniaturization of the motes.

The actuation circuit uses a 2N2222 NPN transistor to actuate the high-power relay via a pulse signal from the MCU. A 470 µf capacitor is used to mitigate the wear of the mechanical closing circuit by adding a time delay. A red led indicated the status of the relay(on/off). See a complete schematic and PCB design as well as the manufactured and assembled motes for the Arduino and ESP32 based shield in Appendix A.

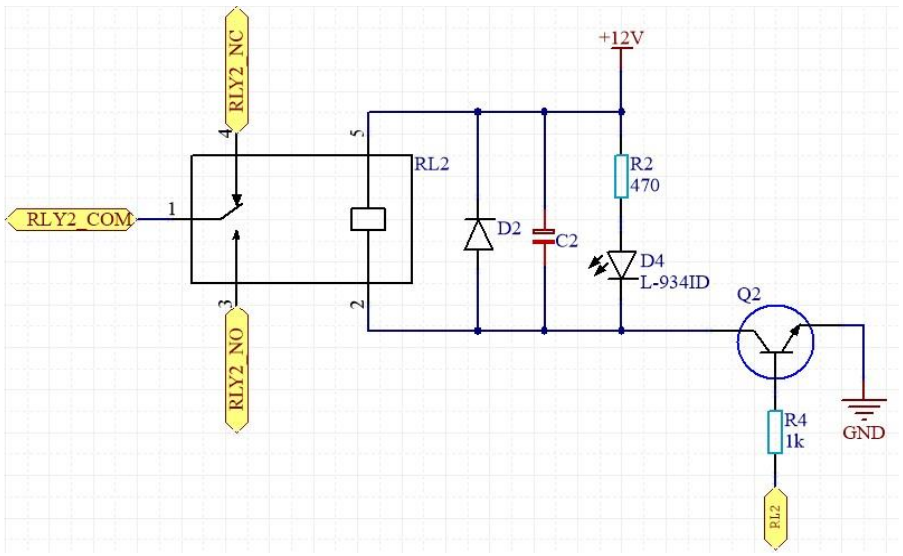


Figure 4.4: Schematic describing the actuation within the mote shield

The Final schematics used to design and manufacture the PCB(s) for the sensing and actuating shields (Appendix B, figure B.1) are shown in Appendix A, figure A.1 and A.2. The PCB were design on Altium designer 17 using the two layers (top and bottom) method easing routing and enabling larger track size for power lines. This structure enables isolation of current and voltage signals from noise (High power relays) while allowing for faster and better heat dissipation from active components. The Arduino based PCB shield were physically cut out to stack up above an AVR or ARM board. All components are DIP (through holes) as the component were to be mounted and soldered by the researcher.

4.3. Platform system integration

Before discussing the flow of information within the experimental platform, the final makeup of the power sensing and actuation motes within the platform is discussed.

4.3.1. Platform motes Hardware Integration

Figure 4.5 shows the different components of each type of motes used in the experimental platform in the HAN. On the left is the Arduino-based mote integrating the relays for actuation, the AC-AC adapter, and the Audio plug for current and voltage measurement on the data acquisition shield. On the right is the ESP32 based mote using an integrating Audio plug for current sensing. This mote can control up to two appliances with a current rating of up to 20A AC.

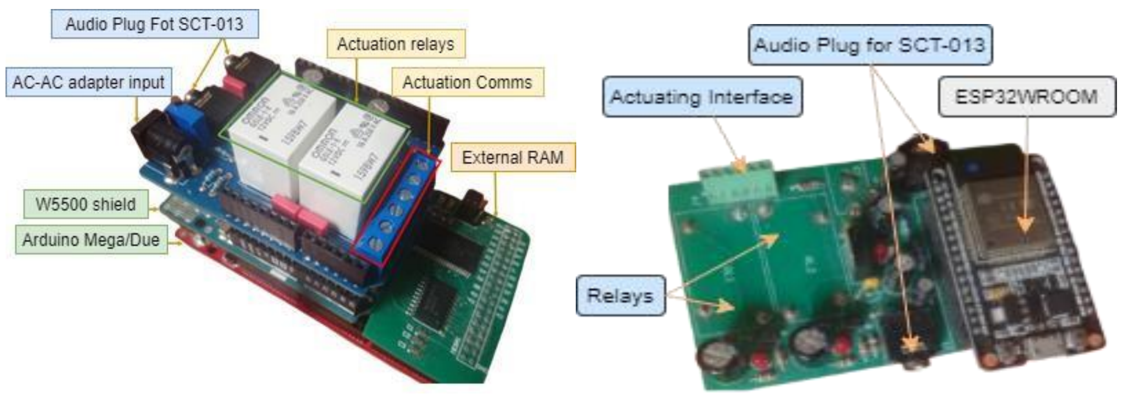


Figure 4.5: Final assembly and main component for platform mote, left) Arduino based mote, right) ESP32 based mote

4.3.2. Sensor Calibration and Signal conditioning

Before deploying the mote, a calibration process was followed to generate the graphs in the figure below for the current signal read from the SCT-013 and a voltage signal from the AC/AC adapter. The raw signal from the SCT-013 is read as a sinusoidal half-wave (yellow line). The RMS value from equation (1) is computed after calibrating the input using equation (4.5) used to compute the conditioned current signal.

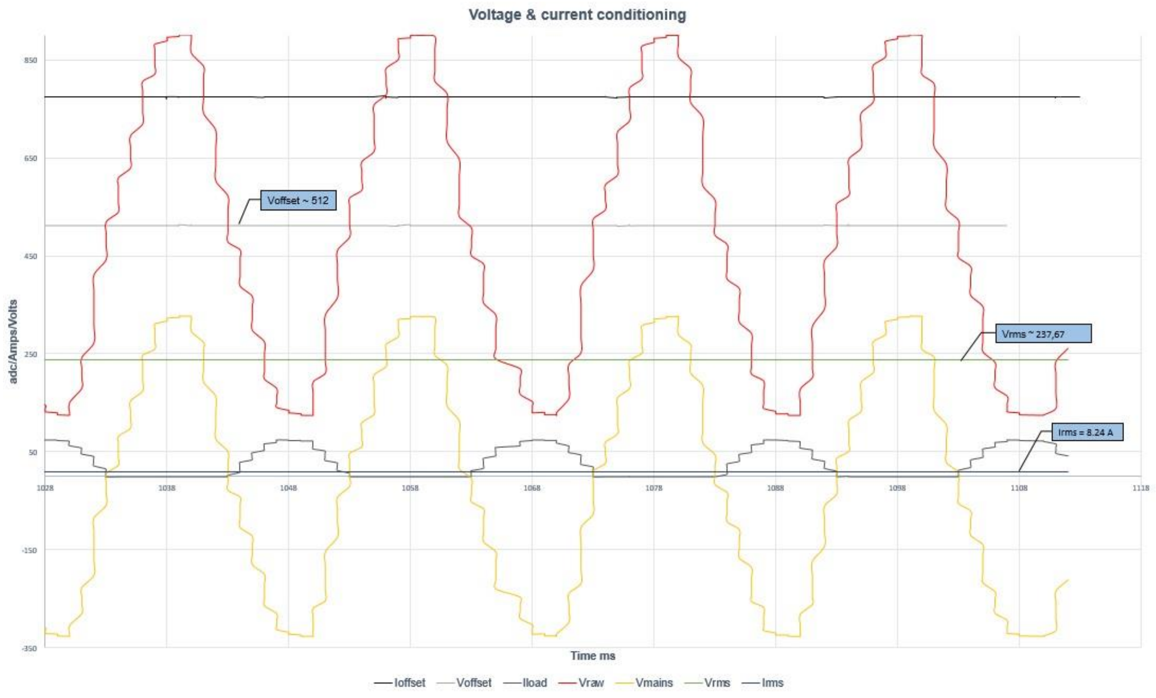


Figure 4.6: Voltage and current signal calibration and conditioning

The discretized ADC reading from the CT sensor signal is used to condition and filter out the actual instant sample value. Based on works from (Learn | OpenEnergyMonitor, 2021) equation (4.5) and (4.6) were formulated. Here, counts represent the MCU ADC equivalent reading for the current signal, $Offset_{counts}$ is the counts equivalent to the DC offset (2.5V DC) that is subtracted to center the waveform around 0V. V_{ref} represent the MCU ADC reference voltage. This value is measured for each type of MCU (4.984V for

Arduino AVR and ~3.3V for DUE and ESP32). ADC_{counts} is the MCU ADC full counts (1024 for AVR and 4098 for DUE and ESP32). A/V is the CT conversion ratio (30 $amps/volt$). I_{rms} calculated from equation (4.1) is thus the CT primary coil current. That is the connected appliance consumption.

$$I_{mains} = (counts - Offset_{counts}) \times \frac{V_{ref}}{ADC_{counts}} \times A/V \tag{4.5}$$

The red curve in figure 4.6, represents the AC/AC adapter input at the MCU ADC. As shown in this figure the sinusoidal full wave is centered about the DC offset which read 512 counts (see grey curve). However, this value is initially sampled by disconnecting the AC/AC adapter socket from the ADC interface and stored as a constant $offset_{counts}$.

$$V_{mains} = (counts - offset_{counts}) \times \frac{V_{ref}}{ADC_{count}} \times Calib_{const} \tag{4.6}$$

$$Calib_{count} = V_{maindef} \times \frac{V_{gain}}{V_{calib}} \tag{4.7}$$

The Voltage signal from the AC/AC step-down adapter is sampled and converted to useful units based on equations (4.6) and (4.7). In (4.7), the ADC counts are filtered by removing the DC offset (~ 2.5V or 512 ADC counts). to convert the digital value to the analog voltage on the mains a calibration constant $Calib_{count}$ is calculated in (4.8). $V_{maindef}$ or 230 Vac in South Africa. V_{gain} is the voltage divider gain (see figure 4.2.1-3)? V_{calib} is the output of the AC/AC adapter (secondary coil voltage) measured using a digital multimeter to 14.6 Vac. The brown graph in figure 4.5 represents the V_{mains} centered around 0 V after conditioning and the V_{rms} is calculated as 237.67 V.

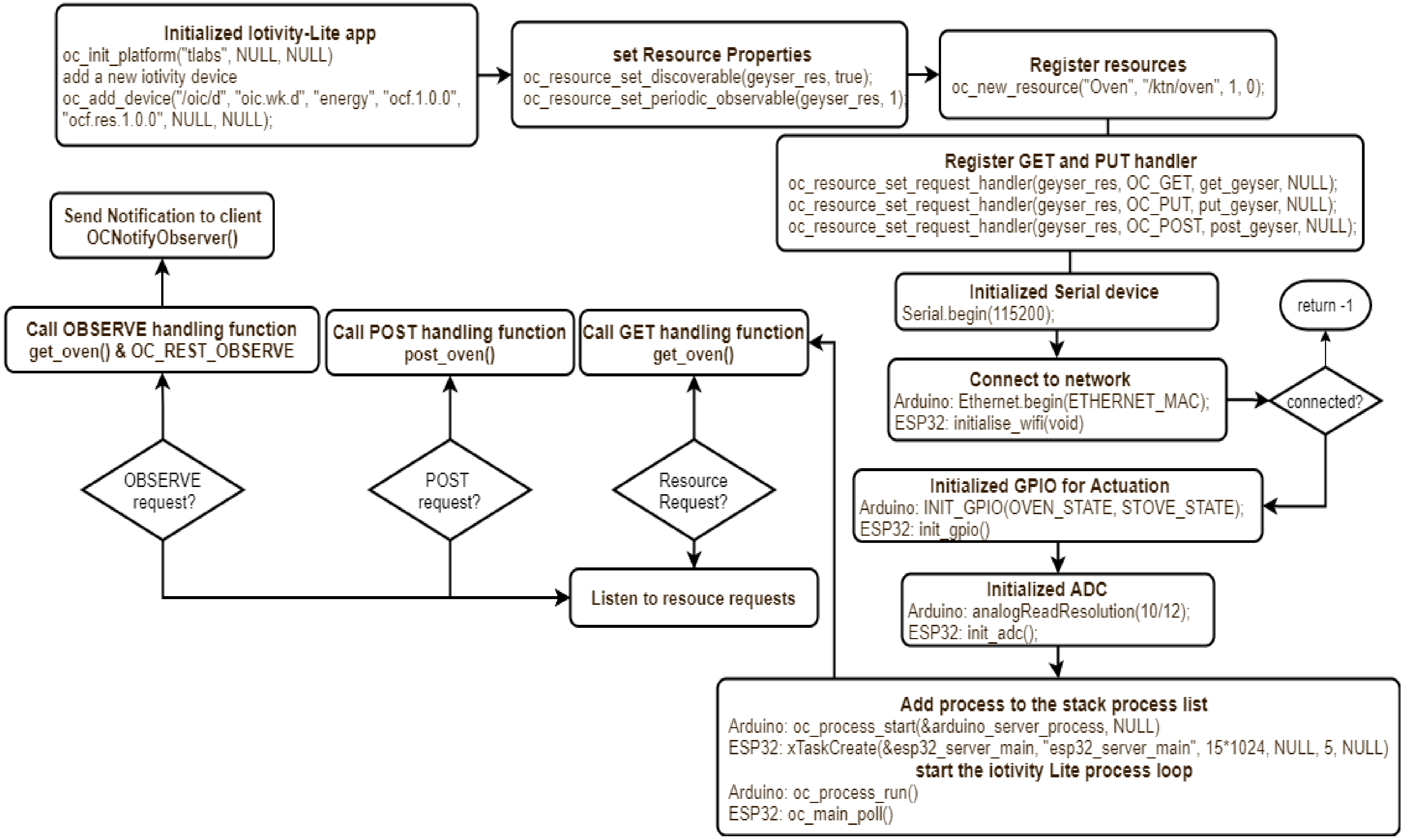
4.3.3. Firmware development for Platform motes.

The firmware running on the HAN devices (Arduino and ESP32) is composed of the IoTivity-Lite server code and the low-level sensing code interfacing to the device's ADC and GPIO registers to control the mote actuation devices. The low-level code for interfacing to the sensing and actuating circuit implement equation (4.1) to (4.8) to compute the power properties of related appliances.

This code is used by the higher-level server code within the GET and PUT methods. Figure 4.7 depicts the algorithm for the DAQ system. The power properties for an appliance connected to either a mote based on the Arduino or ESP32 MCU were computed based on the algorithm in Figure D-1 in Appendix D. The algorithm samples the current and voltage for 25 cycles (at 50Hz) or 500ms to calculate the different root means square properties and accumulate those to calculate the power consumption. The Arduino AVR use a 10bits ADC setting while ARM and ESP32 boards used 12bits resolution.

The computed data is stored in a structure that will be made available for the IoTivity server code. The code implementing this algorithm was developed in C language and a complete listing can be seen in Appendix D. the IoTivity server that runs on either MCU architecture listens to CRUDN from the user to monitor and update the connected appliances. Figure 4.7 depicts the flow of operation of the server code from initialization to servicing of resources requests within the local HAN. See a complete listing of the IoTivity server code for Arduino and ESP32 in Appendix C. Network configuration and cloud BaaS communication. Network configuration and communication flow within the experimental platform are presented in figure 4.13. The communication is layered around a HAN, a gateway, a cloud BaaS. It leverages network (WIFI and Ethernet) infrastructure in the residential place to provide wireless communication. All HAN devices have statically assigned IP addresses on the private network 192.168.0.X starting at 192.168.0.100 for the Raspberry PI gateway. When powered on, two servers run on the gateway. First is the IoT-rest-API-server that provides a REST API over HTTP or HTTPS access to devices on the IoTivity HAN.
Secondly, is the “main server” that initialized connectivity to the cloud BaaS. However, the platform cloud interface first creates using the Back4App free plan a BaaS App here “IoTivitySmartApp” (figure 4.9). this free plan offers file storage up to 1GB and real-time database storage of up to 0.25 GB on a hosted Mango DB (Back4App, 2017). The plan limit databases query to 10 requests/second and allow 10K request per month limited the data throughput within the platform. However, this is enough to demonstrate the capabilities proposed by this solution. Thus, on the Back4App App is created the “Loads”, “SmartHomes”, ”DRM” classes to host the data for the energy cloud services the platform offers (see figure 4.10 and 4.12). For this implementation extension to multiple smart homes, the “owning” relationship mechanism implemented amongst the different classes. This mechanism scales the system by enabling many owners

in the “User” class to own a specific “smart home” that owns many different appliances in the “Loads” class and a specific “DRM” service data. Moreover, this method allows us not to create a class/table for each smart home context, thus keeping all related data together, easing development and maintenance of the platform.


Initialized Iotivity-Lite app
add a new iotivity device
set Resource Properties
Register resources
Register GET and PUT handler
Initialized Serial device
Connect to network
connected?
return -1
Initialized GPIO for Actuation
Initialized ADC
Add process to the stack process list
start the iotivity Lite process loop
Listen to resouce requests
Resource Request?
POST request?
OBSERVE request?
Call GET handling function
Call POST handling function
Call OBSERVE handling function
Send Notification to client

```
oc_init_platform("tlabs", NULL, NULL)
oc_add_device("/oic/d", "oic.wk.d", "energy", "ocf.1.0.0", "ocf.res.1.0.0", NULL, NULL);

oc_resource_set_discoverable(geyser_res, true);
oc_resource_set_periodic_observable(geyser_res, 1);

oc_new_resource("Oven", "/ktn/oven", 1, 0);

oc_resource_set_request_handler(geyser_res, OC_GET, get_geyser, NULL);
oc_resource_set_request_handler(geyser_res, OC_PUT, put_geyser, NULL);
oc_resource_set_request_handler(geyser_res, OC_POST, post_geyser, NULL);

Serial.begin(115200);

Arduino: Ethernet.begin(ETHERNET_MAC);
ESP32: initialise_wifi(void)

Arduino: INIT_GPIO(OVEN_STATE, STOVE_STATE);
ESP32: init_gpio()

Arduino: analogReadResolution(10/12);
ESP32: init_adc();

Arduino: oc_process_start(&arduino_server_process, NULL)
ESP32: xTaskCreate(&esp32_server_main, "esp32_server_main", 15*1024, NULL, 5, NULL)

Arduino: oc_process_run()
ESP32: oc_main_poll()

get_oven()
post_oven()
get_oven() & OC_REST_OBSERVE
OCNotifyObserver()
```

Figure 4.7: Algorithm for IoTivity Server

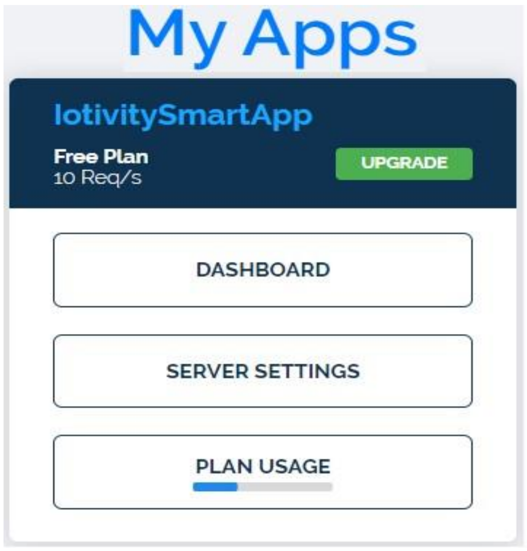
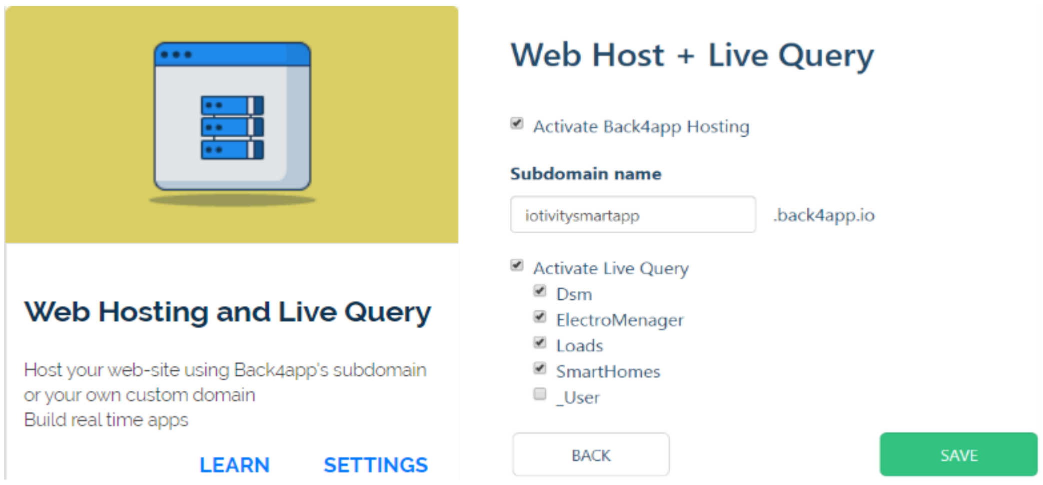


Figure 4.8: Platform App on Back4App BaaS
Adapted from (Back4App, 2017)

After an app was created and the databases classes for a smart home context define following a user login/signup process, the “main server” start a pub-sub subscription to the related “smart home”, DRM and Loads resources on the cloud platform using the Parse Server Live Query mechanism. This connection is realized by authenticated the user and defining the “smart home” against the username that was authenticated. This tool is part of the Back4App BaaS services and is user-configurable by adding the classes (holding database entries/objects) that will be part of the subscription services for the “main server” running on each home gateway and the Energy App developed around the Parse Android API working as a Parse Live Query client (see figure 4.10 below). This tool enables each client endpoint to receive events on the entry in the subscription list. Amongst other events emitted by the Live Query subscription are the “create” and “update” events which are received in real-time by the subscribed client along with meta-data regarding the specific entry that was created/updated. The Parse Live Query mechanism is performant, secure, and easy to use. It leverages the WebSocket technologies with its backend requirements handle by the Back4App cloud infrastructure.

Using the Back4App App application ID, JavaScript Key (for gateway serverclient), and its Master Key for authentication purposes, and the App server URL, the parse Live Query mechanism is initialized. For the platform, the gateway server requirements are implemented using the NodeJS API while the Android API was leverage for the Energy App. An important feature of the Parse server on the Back4App platform is the Cloud code functionality. Cloud Code enables an application developer to offset the application backend computing to the cloud infrastructure. According to Back4App, (2017) this tool enables the developer to run NodeJS functions directly on the Back4App cloud.



Web Hosting and Live Query

Host your web-site using Back4app's subdomain or your own custom domain
Build real time apps

LEARN SETTINGS

Web Host + Live Query

☑ Activate Back4app Hosting

Subdomain name

iotivitysmartapp .back4app.io

☑ Activate Live Query
- ☑ Dsm
- ☑ ElectroMenager
- ☑ Loads
- ☑ SmartHomes
- ☐ _User

BACK SAVE

Figure 4.9: Back4App Parse Live Query Tool Configurations
Adapted from (Back4App, 2017)

The code is executed either via API or SDK call on the Energy App or gateway server. The developer can upload JavaScript functions to the Back4App cloud code server after deploying (figure 4.11). This step immediately makes the Cloud Code Functions available to the IoT platform and can be used to implement services for the platform connecting to third-party tools. See Appendix A for the cloud code function “main.js” deployed on the platform.

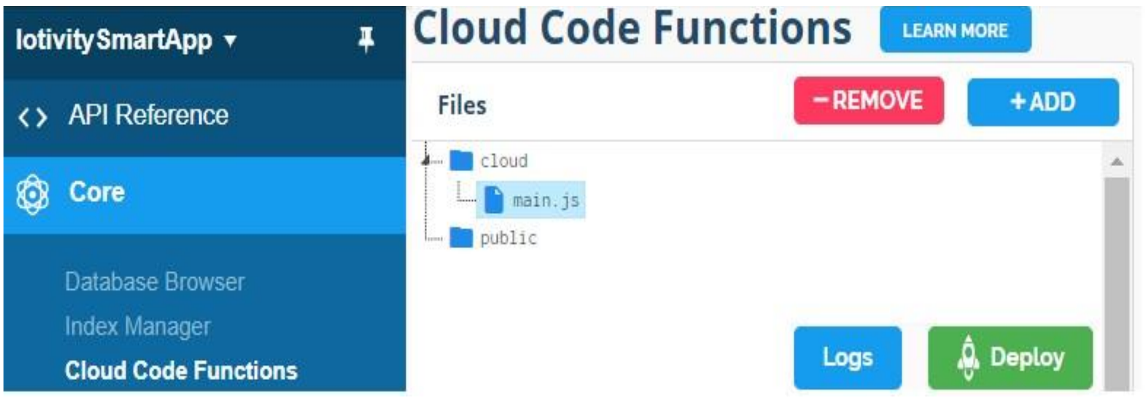
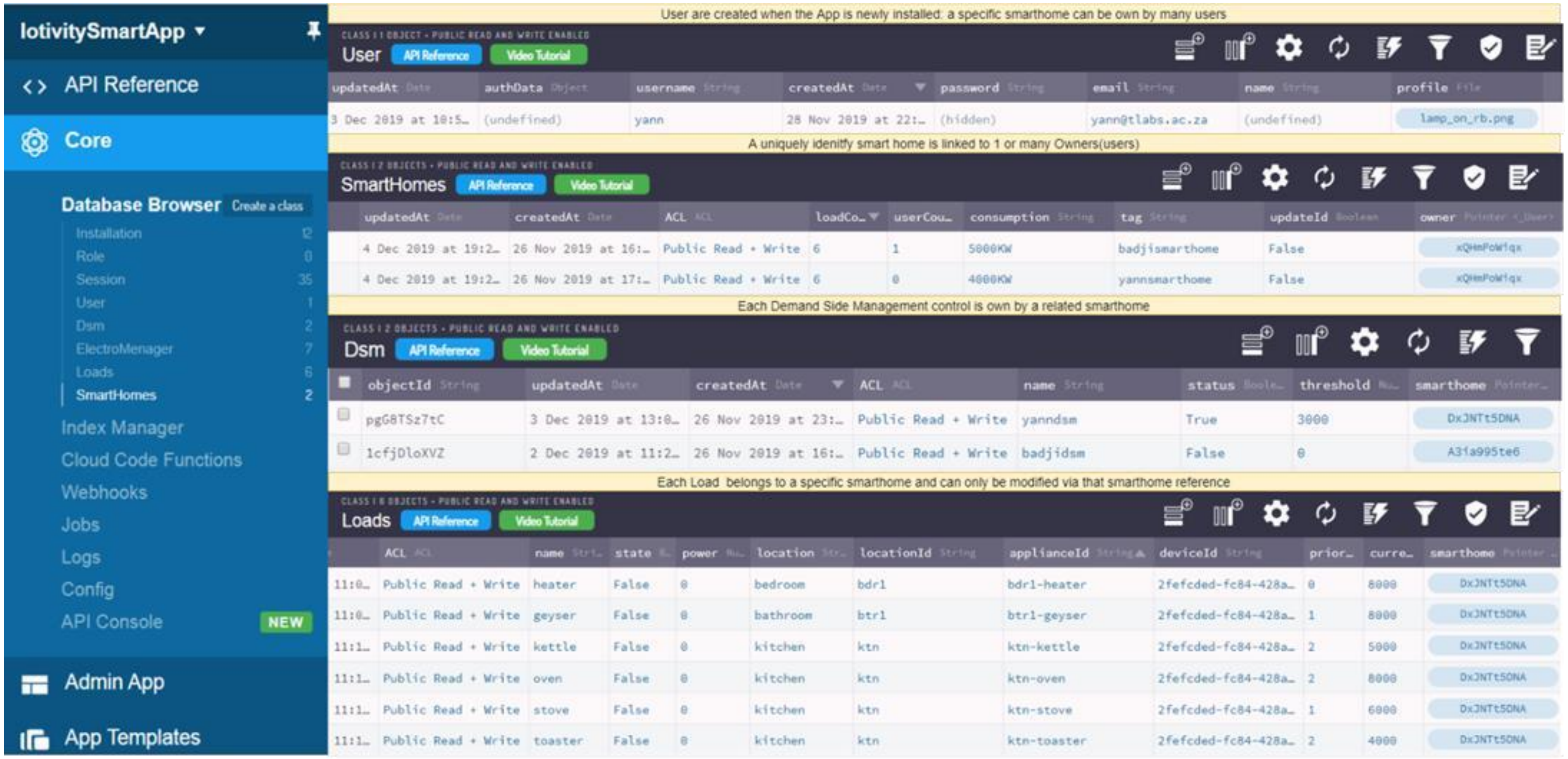


Figure 4.10: Configuring and Uploading Cloud Code Functions

Once the cloud storage ad computing tools have been configured, the gateway server initiates a login/signup sequence with the cloud user authentication services. This process confirms the user and creates on the gateway side the smart home identification based on the credentials username. From this step, the Live Query mechanism is configured and initiate the server based on the communication flow of figure 4.13. The gateway server providing the cloud interface connects to the Parse server and initiates a secure Live Query client based on the Apps ID and JavaScript keys. This is followed by a subscription open for the “SmartHomes”, “Loads”, and “DRM” classes resources for the “create”, “update” events. Next, the gateway server initiates the local DBs (monitor and Loads tables are created if not existing already) for offline storage (MySQL DB engine). The offline and offline storage is synchronized with an initial DB query to the parse server which returns the provisioned number of loads. This process is two-fold.

First, the gateway server sends a DB query to get the number of known and provisioned appliances and retrieved those from the local storage which is also able to store newly discovered appliances. Secondly, a resource discovery from the is sent to the IoT-rest-API-server which generates a multicast request on the IoTivity COAP network to retrieve all resources. Subsequently, each appliance in the local DB is updated after submitting GET requests for their properties (state, power, and current). Lately, the remote DB appliance properties are also updated. These initial steps aim to update the resource IDs as those are reset after power failure. After initialization, an observation service based on pub-sub to resources on the IoTivity network using the OBSERVE mechanism is started and resources properties are regularly updated (although an update threshold is added) considering that cloud transactions are expensive for Free plan offer on Back4App. When a mobile client using the energy, App participates in the platform information exchange, first, the App establishes a connection to the cloud backend platform and starts a client subscription after the client Is successfully login/signup as an authenticated user. Back4App BaaS security features is implemented on the client and on the home gateway side to provide a secured data communication in both ways. As depicted in figure 4.14, all GET requests are submitted as Parse GET queries for each mobile client to access the related homes databases. a PUT request is forwarded to the Parse server on the cloud platform. As the gateway server is in Live Query mode, those requests are received as update events on the resources (here loads/appliances). The gateway server thus generates an HTTP POST request to the IoT-rest-API-server which generates a CoAP POST (i.e., */apil/oic/ktn/kettle?di=’’)* with the new state (i.e., state: true/false) which thus turn the corresponding appliance on/off. Using the Parse Live query mechanism(observation), the smartphone App listens to updates on appliances' power properties from the gateway server’s observer service and updates the App front-end.



User are created when the App is newly installed: a specific smarthome can be own by many users

CLASS | 1 OBJECT • PUBLIC READ AND WRITE ENABLED

User

updatedAt Date	authData Object	username String	createdAt Date	password String	email String	name String	profile File
3 Dec 2019 at 10:5…	(undefined)	yann	28 Nov 2019 at 22:…	(hidden)	yann@tlabs.ac.za	(undefined)	lamp_on_rb.png

A uniquely idenitfy smart home is linked to 1 or many Owners(users)

CLASS | 2 OBJECTS • PUBLIC READ AND WRITE ENABLED

SmartHomes

updatedAt Date	createdAt Date	ACL ACL	loadCo…	userCou…	consumption String	tag String	updateId Boolean	owner Pointer <_User>
4 Dec 2019 at 19:2…	26 Nov 2019 at 16:…	Public Read + Write	6	1	5000KW	badjismarthome	False	xQHmPoWiqx
4 Dec 2019 at 19:2…	26 Nov 2019 at 17:…	Public Read + Write	6	0	4000KW	yannsmarthome	False	xQHmPoWiqx

Each Demand Side Management control is own by a related smarthome

CLASS | 2 OBJECTS • PUBLIC READ AND WRITE ENABLED

Dsm

objectId String	updatedAt Date	createdAt Date	ACL ACL	name String	status Boole…	threshold Nu…	smarthome Pointer…
pgG8TSz7tC	3 Dec 2019 at 13:0…	26 Nov 2019 at 23:…	Public Read + Write	yanndsm	True	3000	DxJNTt5DNA
1cfjDloXVZ	2 Dec 2019 at 11:2…	26 Nov 2019 at 16:…	Public Read + Write	badjidsm	False	0	A31a995te6

Each Load belongs to a specific smarthome and can only be modified via that smarthome reference

CLASS | 6 OBJECTS • PUBLIC READ AND WRITE ENABLED

Loads

	ACL ACL	name Stri…	state B…	power Nu…	location Str…	locationId String	applianceId String	deviceId String	prior…	curre…	smarthome Pointer…
11:0…	Public Read + Write	heater	False	0	bedroom	bdr1	bdr1-heater	2fefcded-fc84-428a…	0	8000	DxJNTt5DNA
11:0…	Public Read + Write	geyser	False	0	bathroom	btr1	btr1-geyser	2fefcded-fc84-428a…	1	8000	DxJNTt5DNA
11:1…	Public Read + Write	kettle	False	0	kitchen	ktn	ktn-kettle	2fefcded-fc84-428a…	2	5000	DxJNTt5DNA
11:1…	Public Read + Write	oven	False	0	kitchen	ktn	ktn-oven	2fefcded-fc84-428a…	2	8000	DxJNTt5DNA
11:1…	Public Read + Write	stove	False	0	kitchen	ktn	ktn-stove	2fefcded-fc84-428a…	1	6000	DxJNTt5DNA
11:1…	Public Read + Write	toaster	False	0	kitchen	ktn	ktn-toaster	2fefcded-fc84-428a…	2	4000	DxJNTt5DNA

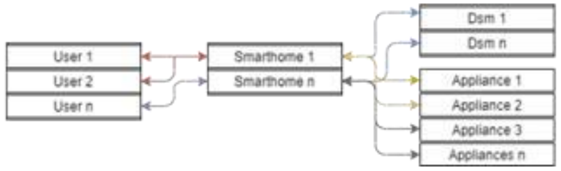
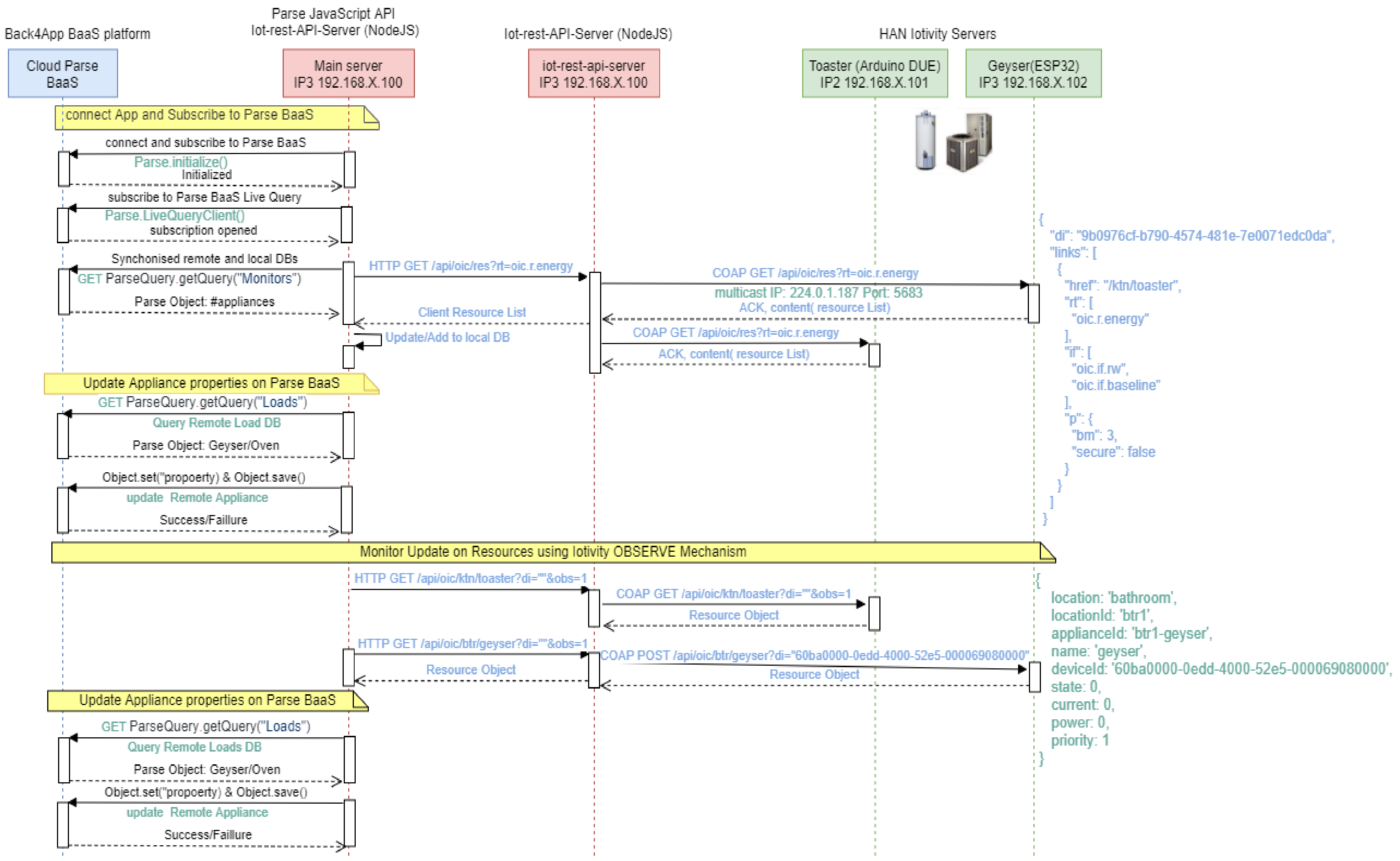


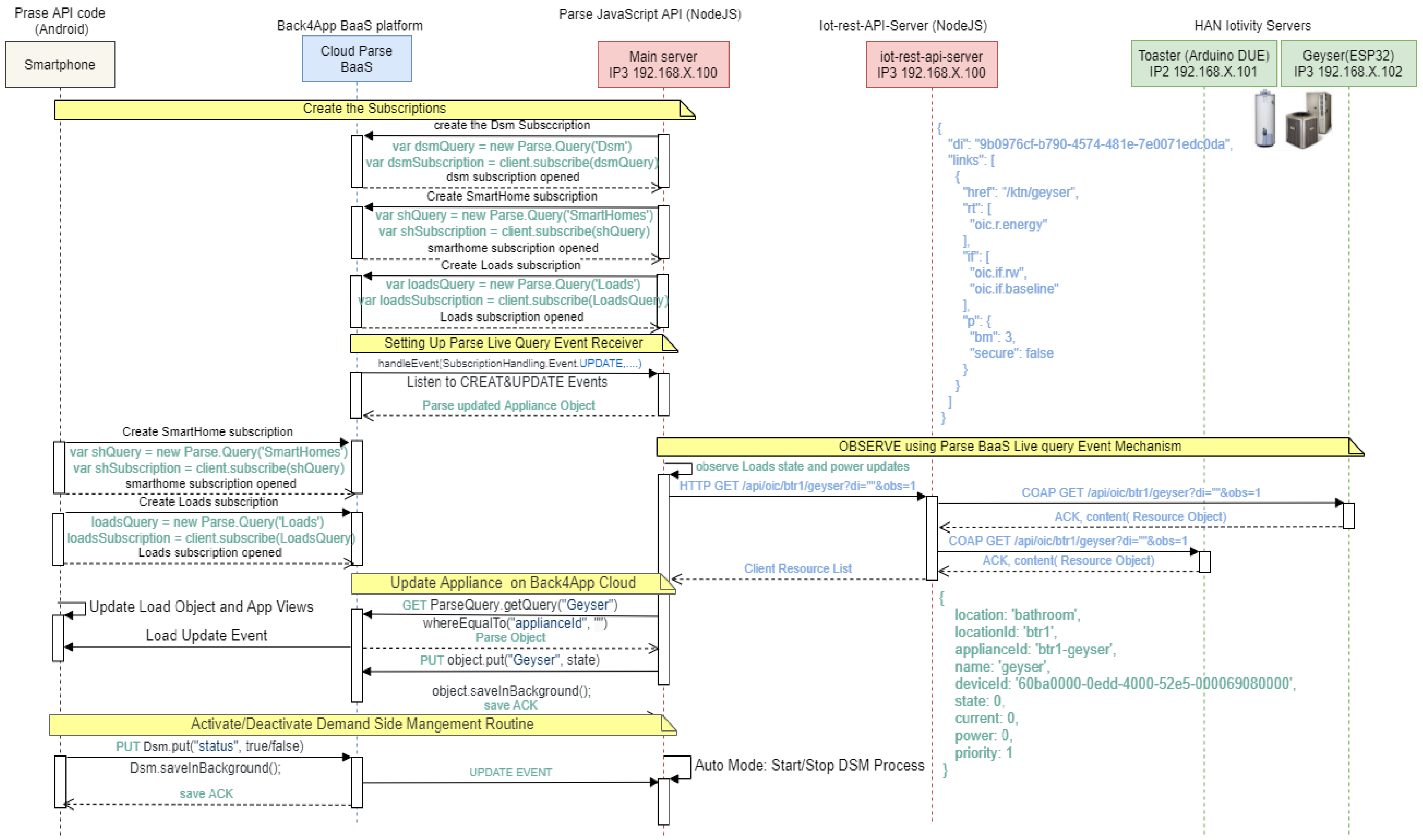
Figure 4.11: Platform databases structure and connections on Back4App BaaS


Back4App BaaS platform
Parse JavaScript API
Iot-rest-API-Server (NodeJS)
Iot-rest-API-Server (NodeJS)
HAN Iotivity Servers
Cloud Parse BaaS
Main server IP3 192.168.X.100
iot-rest-api-server IP3 192.168.X.100
Toaster (Arduino DUE) IP2 192.168.X.101
Geyser(ESP32) IP3 192.168.X.102
connect App and Subscribe to Parse BaaS
connect and subscribe to Parse BaaS
Parse.initialize()
Initialized
subscribe to Parse BaaS Live Query
Parse.LiveQueryClient()
subscription opened
Synchonised remote and local DBs
GET ParseQuery.getQuery("Monitors")
Parse Object: #appliances
HTTP GET /api/oic/res?rt=oic.r.energy
COAP GET /api/oic/res?rt=oic.r.energy
multicast IP: 224.0.1.187 Port: 5683
ACK, content(resource List)
Client Resource List
Update/Add to local DB
COAP GET /api/oic/res?rt=oic.r.energy
ACK, content(resource List)
Update Appliance properties on Parse BaaS
GET ParseQuery.getQuery("Loads")
Query Remote Load DB
Parse Object: Geyser/Oven
Object.set("propoerty) & Object.save()
update Remote Appliance
Success/Faillure
Monitor Update on Resources using Iotivity OBSERVE Mechanism
HTTP GET /api/oic/ktn/toaster?di=""&obs=1
COAP GET /api/oic/ktn/toaster?di=""&obs=1
Resource Object
HTTP GET /api/oic/btr/geyser?di=""&obs=1
COAP POST /api/oic/btr/geyser?di="60ba0000-0edd-4000-52e5-000069080000"
Resource Object
Resource Object
Update Appliance properties on Parse BaaS
GET ParseQuery.getQuery("Loads")
Query Remote Loads DB
Parse Object: Geyser/Oven
Object.set("propoerty) & Object.save()
update Remote Appliance
Success/Faillure

```
{
  "di": "9b0976cf-b790-4574-481e-7e0071edc0da",
  "links": [
   {
    "href": "/ktn/toaster",
    "rt": [
     "oic.r.energy"
    ],
    "if": [
     "oic.if.rw",
     "oic.if.baseline"
    ],
    "p": {
     "bm": 3,
     "secure": false
    }
   }
  ]
 }
```

```
{
  location: 'bathroom',
  locationId: 'btr1',
  applianceId: 'btr1-geyser',
  name: 'geyser',
  deviceId: '60ba0000-0edd-4000-52e5-000069080000',
  state: 0,
  current: 0,
  power: 0,
  priority: 1
}
```

Figure 4.12: Network configuration and communication in the Platform


Prase API code (Android)
Smartphone
Back4App BaaS platform
Cloud Parse BaaS
Parse JavaScript API (NodeJS)
Main server IP3 192.168.X.100
Iot-rest-API-Server (NodeJS)
iot-rest-api-server IP3 192.168.X.100
HAN Iotivity Servers
Toaster (Arduino DUE) IP2 192.168.X.101
Geyser(ESP32) IP3 192.168.X.102
Create the Subscriptions
create the Dsm Subsccription
var dsmQuery = new Parse.Query('Dsm')
var dsmSubscription = client.subscribe(dsmQuery)
dsm subscription opened
Create SmartHome subscription
var shQuery = new Parse.Query('SmartHomes')
var shSubscription = client.subscribe(shQuery)
smarthome subscription opened
Create Loads subscription
var loadsQuery = new Parse.Query('Loads')
var loadsSubscription = client.subscribe(LoadsQuery)
Loads subscription opened
Setting Up Parse Live Query Event Receiver
handleEvent(SubscriptionHandling.Event.UPDATE,....)
Listen to CREAT&UPDATE Events
Parse updated Appliance Object
Create SmartHome subscription
var shQuery = new Parse.Query('SmartHomes')
var shSubscription = client.subscribe(shQuery)
smarthome subscription opened
Create Loads subscription
loadsQuery = new Parse.Query('Loads')
loadsSubscription = client.subscribe(LoadsQuery)
Loads subscription opened
OBSERVE using Parse BaaS Live query Event Mechanism
observe Loads state and power updates
HTTP GET /api/oic/btr1/geyser?di=""&obs=1
COAP GET /api/oic/btr1/geyser?di=""&obs=1
ACK, content(Resource Object)
COAP GET /api/oic/btr1/geyser?di=""&obs=1
ACK, content(Resource Object)
Client Resource List
Update Appliance on Back4App Cloud
GET ParseQuery.getQuery("Geyser")
whereEqualTo("applianceId", "")
Parse Object
PUT object.put("Geyser", state)
object.saveInBackground();
save ACK
Update Load Object and App Views
Load Update Event
Activate/Deactivate Demand Side Mangement Routine
PUT Dsm.put("status", true/false)
Dsm.saveInBackground();
save ACK
UPDATE EVENT
Auto Mode: Start/Stop DSM Process

```
{
  "di": "9b0976cf-b790-4574-481e-7e0071edc0da",
  "links": [
    {
      "href": "/ktn/geyser",
      "rt": [
        "oic.r.energy"
      ],
      "if": [
        "oic.if.rw",
        "oic.if.baseline"
      ],
      "p": {
        "bm": 3,
        "secure": false
      }
    }
  ]
}
```

```
{
  location: 'bathroom',
  locationId: 'btr1',
  applianceId: 'btr1-geyser',
  name: 'geyser',
  deviceId: '60ba0000-0edd-4000-52e5-000069080000',
  state: 0,
  current: 0,
  power: 0,
  priority: 1
}
```

Figure 4.13: communication flow with Android smartphone App

4.4. Chapter Summary

In this chapter was described the equipment used for the case study addressing the thesis research objectives. The focus was on the design of the motes used as IoTivity resources servers in the HAN. Here was considered the mote’s hardware and software requirements, mainly the IoTivity firmware based on the developed Arduino port. The DAQ interface that gathers the CT and controls the connected appliances, presenting the signal conditioning and conversion equations was discussed. Having described the hardware, consideration was given to the high-level software that interacts with the Back4App cloud service, REST service, and discussed the backend interaction from the resource’s server in the IoTivity HAN to the back4App cloud and the Smartphone Energy App developed.

CHAPTER FIVE
CASE STUDY, RESULTS, AND DISCUSSION

5.1. Introduction

To illustrate the value and performance of the platform, this chapter proposed a case study deploying several scenarios demonstrating the ability of this platform to handle certain applications about energy management. Lately, the results are presented, and a thorough discussion is carried on the different observations.

5.2. Energy Monitoring

The objective here is to provide granular feedback at the appliance level anytime and anywhere via an engaging graphical display on an Android smartphone. The goal is to better understand the demand characteristic of the home under consideration which will help to make smart decisions about controlling its demand. Therefore, the work aims to build energy literacy in residential places and demonstrate energy efficiency based on the IoT "mindchanging" feedback. Furthermore, this can be used to bring consumer visibility to phantom load from unmanageable appliances (i.e., TV, PC, phone chargers, etc.). The performance is measured in the ability of the energy App to provide users with appliance consumption and status information as well as granular and overall energy consumption of a home in a near real-time manner.

5.3. Home Automation

In this experimental test, the IoTivity platform works as a home automation service enabling users to control the status of their appliances via the energy App by turning the appliance on/off anytime and anywhere to manually reduce their energy usage. Performance is observed in the ability to change an appliance status in a near real-time manner as well as updating the user interface with the appliance's new status.

5.4. Demand Response Management

DRM algorithm for peak load management was implemented a as a service that aims to demonstrate the impact of the IoTivity platform on residential load efficiency. The research followed related works in the area of HEM to define the experimental model. To demonstrate the performance of their IoT architecture for residential load, Viswanath et al., (2016), implemented an experiment based on a maximum allowable peak threshold of 33KW. In their work, the DRM control was notified each time the overall consumption was greater than 33.5KW. The author's algorithm control light bulbs at each house in peak time by slotting a 24 hours’ time duration into 8640-time periods. That is their smart grid simulation detected the total demand every 10 seconds. *Al Faruque & Vatanparvar, (2016)*, implemented smart transformer Control-as-a-service over fog computing limiting the load of each home at 4KW. The authors, limiting the maximum load of microgrid transformer alimenting a group of homes at 20KW. The algorithm monitors the status of the power source and activates a DR signal when overload by cycling all home and shedding load in a home that has exceeded the 4KW thresholds. In both studies, the demonstrated DRM does not consider user preferences. Rasheed et al., 2016, introduce three-level priority scheduling for home appliances so users can switch on home appliances subject to their satisfaction level and preferences. That is, consumers willing to turn on appliances immediately could allocate it a higher priority and vice versa. Peak load DRM depends on mathematical models. In this case study, attention is given to regularly operated or fixed appliances and develop an algorithm based on equations (5.1) to (5.4) adapting formulation from (Hussain *et al.*, 2018) and (Khan *et al.*, 2019). Therefore, the experimental DRM is the home automation system application relying on the IoT granular feedback (from energy monitoring) to handle via a priority-based load shedding program home appliances load. The experimentation focusses on powerintensive appliances (70% of domestic consumption) with constant consumption (resistive loads) and maximum rating and priority as shown in Table 5.1.

Table 5.1: Appliances in the considered home with their typical priority level Adapted from: (Qayyum et al., 2015)

Home Appliances	**Maximum Rating(W)**	**Priority**	
		Morning	**Evening**
Electric geyser	3000	High	Low
Kettle	2200	Medium	Medium
Toaster	950	High	Low
Oven	2350	Low	High
Stove	3000	Medium	High
Iron	1800	Medium	Low

A load cycling (load reduction) system is implemented to shutdown appliances' operation based on user-defined consumption thresholds or consumption goals. Priority is introduced in this case study to dynamically limit the home instant load. However, a default value of 5KW based on literature and the appliance of table 5.1 will be used by the algorithm that will be implemented. Figure 5.1 depicts the experimental platform and the data flow for this case study.

5.4.1. Energy Consumption Model

A home Load primarily from resistive load appliances is considered. For the case study model, let $A_n \epsilon \{a_1, a_2, a_3, . . ., a_n\}$, such that $a_1, a_2, a_3, . . ., a_n$ represents each appliance. The model considers 6 appliances (table 5.1). The peak periods in the south African context are two. The morning peak is from 6 am to 9 am while the evening peak is from 6 pm to 9 pm. In the model each peak period is slice into a horizon time slot series T ϵ {1, 2, 3, . . ., T}. since each peak period span the same time length of τ_{peak} (4 hours), considering that each time slot is 15 min long, thus T is series of 16 elements. The total power consumption during a peak period is expressed as $\boldsymbol{\varsigma_{A_n T_L}}$

$$\varsigma_{A_n T_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \zeta(t)\right) \qquad (5.1)$$

Where $\boldsymbol{P_n(t)}$ is the power consumption for appliance a_n at time slot t ϵ T. $\boldsymbol{\zeta(t)}$ ϵ [0,1] is the operational state of appliances in time interval t ϵ T. Similarly, the total cost per peak period of the A_n,

$$£_{A_n T_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \varepsilon(t) \times \zeta(t)\right) \qquad (5.2)$$

Where $\varepsilon(t)$ represent the cost of electricity at time t ϵ T.

Based on equation (1), the DRM algorithm for the case study is formulated as below

$$\varsigma_{A_n T_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \zeta(t)\right) \leq \gamma(t) \qquad (5.3)$$

Where $\gamma(t)$is the home threshold? That is $\gamma(t)$ is the maximum allowable peak load at time t ϵ T. A dynamic value for $\gamma(t)$ of 5 KW is used.

5.4.2. DRM Algorithm implementation

Each home DRM run on the raspberry PI gateway. The DRM algorithm implements equations (5.1) to (5.3) digitized within the algorithm, looping through the list of appliances and calculated the total power consumption for time slot t and storing that in power series P_n (5.5) which is a list of $б_n$ elements (5.4). At the exit of a peak period, the algorithm thus calculates the total power consumption, the average peak power (5.6), and the maximum power consumption (4.14) for each peak period.

$$\tau_{peak} \times 3600\ б_n = \underline{\qquad\qquad}900 \qquad (5.4)$$

$$б^n$$

$$P_{avg} = \sum\underline{\qquad\qquad}_1 \quad P_{peak}$$

$$(5.5)\ б_n$$

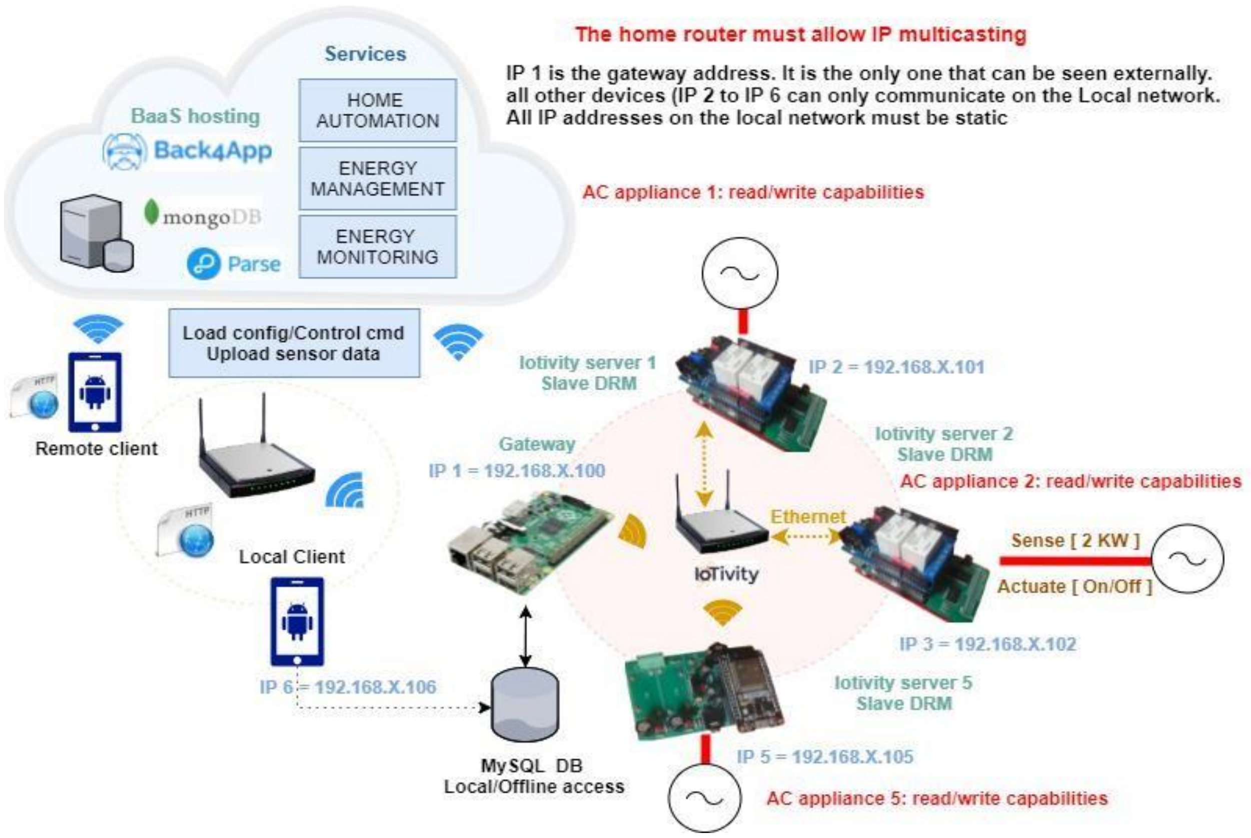
With $P_{\mathrm{peak}} \epsilon\ [P_1, P_2, P_3, \ldots, P_n]$ with $\mathrm{n} \leq б_n$, the max peak power is computed using equation (4.14) below.

$$P_{max} = \max_{n \leq б_n}(P_{peak}) \qquad (5.6)$$

The algorithm is implemented with an electricity price per unit considering a household in the research context (here the city of Cape Town) with consumption equal to or above 600 KWh/month municipality regulated unit at 278.46 c/kWh (City of Cape Town, 2019). Figure C.1 in Appendix C is the pseudo-code for DRM firmware implemented after digitizing equation (4.8) to (4.10) to maintain the household consumption at any instant t. At the end of each peak period, the algorithm calculates the average power, average energy cost as well as maximum power peak and maximum peak energy consumption. This data is made available as statistical info to each smart home user. The code was implemented in NodeJS and can be seen in Appendix D and E.

5.4.3. DRM Scenario Network communication

For the DRM scenario, the communication platform extends the model of figure 4.12. After the gateway server initialized the connection to the parse server BaaS platform, it opens a client subscription to the “DRM” object related to the user smart home and a listening event is activated on the object. The smartphone Energy App activates/deactivates the DRM mode by setting its status true/false. This event is forwarded via Live Query from the Back4App server to the gateway server, which processes the event and load/unload the DRM service. Moreover, an observer process is loaded at initialization by the gateway server to update the parse online objects with the latest appliance's status (see figure 4.13). This process is based on the IoT-rest-API-server and IoTivity-Lite observe mechanism which will return updates on a resource whose properties have changed (i.e., current or power).



The home router must allow IP multicasting
IP 1 is the gateway address. It is the only one that can be seen externally. all other devices (IP 2 to IP 6 can only communicate on the Local network. All IP addresses on the local network must be static
Services
BaaS hosting
Back4App
HOME AUTOMATION
ENERGY MANAGEMENT
ENERGY MONITORING
mongoDB
Parse
Load config/Control cmd
Upload sensor data
Remote client
Local Client
AC appliance 1: read/write capabilities
Iotivity server 1 Slave DRM
IP 2 = 192.168.X.101
Iotivity server 2 Slave DRM
AC appliance 2: read/write capabilities
Gateway
IP 1 = 192.168.X.100
Ethernet
IoTivity
Sense [2 KW]
Actuate [On/Off]
IP 3 = 192.168.X.102
Iotivity server 5 Slave DRM
IP 6 = 192.168.X.106
MySQL DB Local/Offline access
IP 5 = 192.168.X.105
AC appliance 5: read/write capabilities

Figure 5.1: Case study system architecture

5.5. Results

In this section is presented the results from the experimental case study shown in figure 5.2, focusing on the output from each scenario to establish the platform efficiency and performance in realizing each required functionality. Using the setup in the figure below, the experimentation tests the feedback and home automation scenario within the platform. The response from the IoTivity-Lite HAN server device is presented. That is, the Arduino and ESP32 slaves' response to resources request. Then, the underlining software services handling the smart home local and remote connectivity are discussed. In this regard, are described the different initialization steps via curtailed logs of each of the services running on the raspberry PI local home server. Secondly, feedback results, that is real-time home consumption via enhance visualization anytime anywhere using the energy App is exhibited. Thirdly, home automation is demonstrated using the Energy App to turn home appliances on/off. Lastly, the DRM scenario conditions and assumptions are detailed, and the result of the peak shaving algorithm are shown.

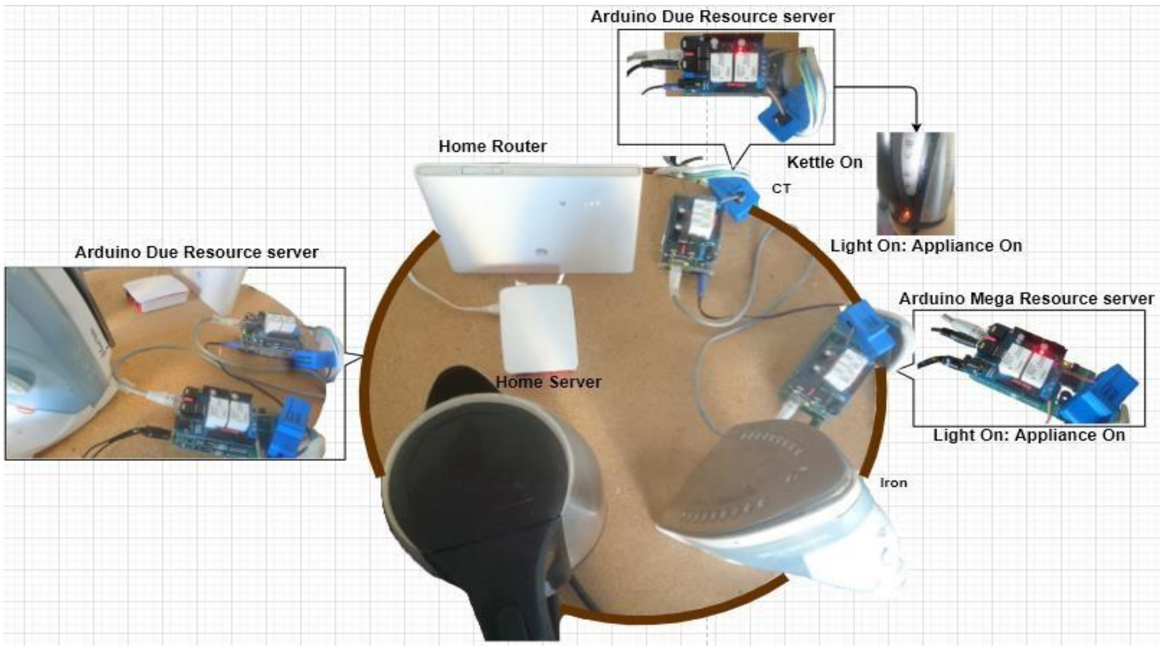
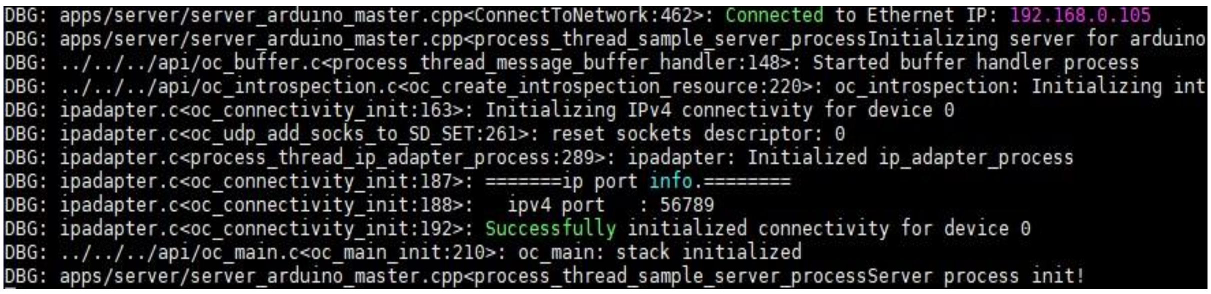

Arduino Due Resource server
Home Router
Kettle On
CT
Light On: Appliance On
Arduino Due Resource server
Arduino Mega Resource server
Home Server
Light On: Appliance On
Iron

Figure 5.2: Experimental Platform used for Scenario’s testing

5.5.1. HAN device responses to GET/POST requests

The firmware on the HAN resource server runs the IoTivity-Lite core that was ported to the AVR and ARM Arduino Arch. In figure 5.3 below are shown the initialization logs for the devices, which request a local IP address within the 192.168.0.1 subnet, initializing the IoTivity core and starting a listening server on IPv4 port 56789 for Arduino devices. The ESP32 slaves use both IPv4 and IPv6 listening sockets as provided by the IoTivity-Lite stack.

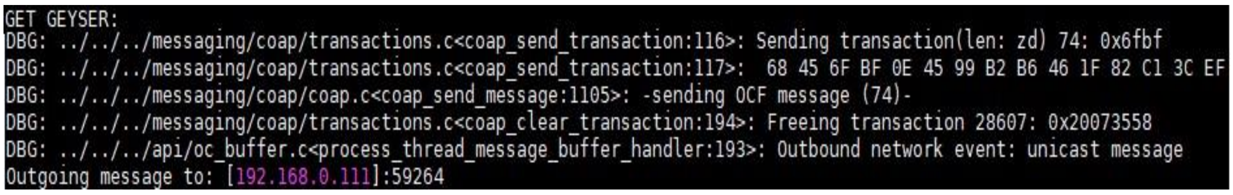


```
DBG: apps/server/server_arduino_master.cpp<ConnectToNetwork:462>: Connected to Ethernet IP: 192.168.0.105
DBG: apps/server/server_arduino_master.cpp<process_thread_sample_server_processInitializing server for arduino
DBG: ../../../api/oc_buffer.c<process_thread_message_buffer_handler:148>: Started buffer handler process
DBG: ../../../api/oc_introspection.c<oc_create_introspection_resource:220>: oc_introspection: Initializing int
DBG: ipadapter.c<oc_connectivity_init:163>: Initializing IPv4 connectivity for device 0
DBG: ipadapter.c<oc_udp_add_socks_to_SD_SET:261>: reset sockets descriptor: 0
DBG: ipadapter.c<process_thread_ip_adapter_process:289>: ipadapter: Initialized ip_adapter_process
DBG: ipadapter.c<oc_connectivity_init:187>: =======ip port info.========
DBG: ipadapter.c<oc_connectivity_init:188>:   ipv4 port   : 56789
DBG: ipadapter.c<oc_connectivity_init:192>: Successfully initialized connectivity for device 0
DBG: ../../../api/oc_main.c<oc_main_init:210>: oc_main: stack initialized
DBG: apps/server/server_arduino_master.cpp<process_thread_sample_server_processServer process init!
```

Figure 5.3: Arduino slave initialization logs

The firmware loaded in all slaves allows these devices to serve the client with resources data handling those as GET/POST requests. The IoT-rest-API-server provisioned devices and resource on the IoTivity-Lite local network after issuing a multicast request on the endpoint (localhost:8000/ioc/res). A client can thus request local resources to issue HTTP requests to the REST server. In figure 5.4, are logs of GET requests received from the slaves, followed by the

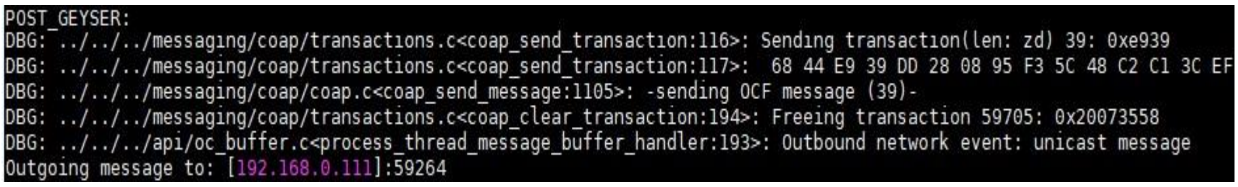
IoTivityLite stack processing of the request and a response (74 bytes of resource data) to the client on 192.168.0.111:59264.



```
GET GEYSER:
DBG: ../../../messaging/coap/transactions.c<coap_send_transaction:116>: Sending transaction(len: zd) 74: 0x6fbf
DBG: ../../../messaging/coap/transactions.c<coap_send_transaction:117>:  68 45 6F BF 0E 45 99 B2 B6 46 1F 82 C1 3C EF
DBG: ../../../messaging/coap/coap.c<coap_send_message:1105>: -sending OCF message (74)-
DBG: ../../../messaging/coap/transactions.c<coap_clear_transaction:194>: Freeing transaction 28607: 0x20073558
DBG: ../../../api/oc_buffer.c<process_thread_message_buffer_handler:193>: Outbound network event: unicast message
Outgoing message to: [192.168.0.111]:59264
```

Figure 5.4:HAN server GET response

A similar POST interaction is executed whenever the client request and update an appliance status (On/Off). In figure 5.5, after updating the state of an appliance from a POST request, the resource server sends a 39 bytes acknowledgment response to the requesting client at 192.168.0.111:8000.

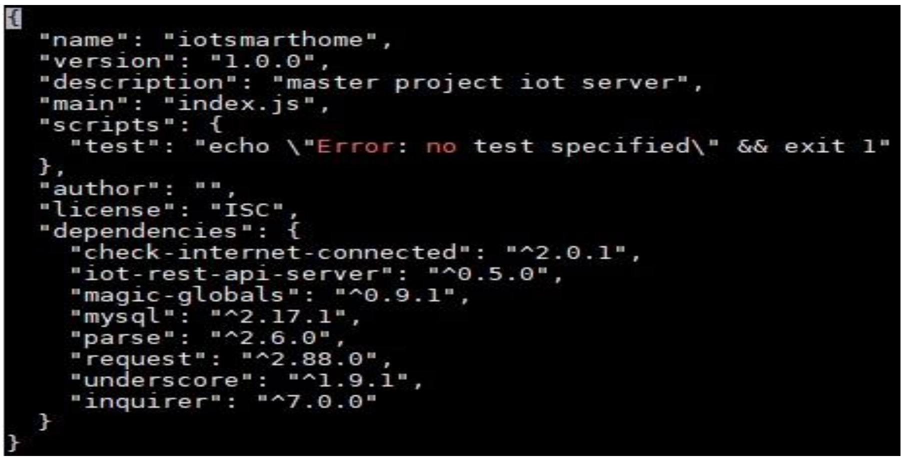


```
POST_GEYSER:
DBG: ../../../messaging/coap/transactions.c<coap_send_transaction:116>: Sending transaction(len: zd) 39: 0xe939
DBG: ../../../messaging/coap/transactions.c<coap_send_transaction:117>:  68 44 E9 39 DD 28 08 95 F3 5C 48 C2 C1 3C EF
DBG: ../../../messaging/coap/coap.c<coap_send_message:1105>: -sending OCF message (39)-
DBG: ../../../messaging/coap/transactions.c<coap_clear_transaction:194>: Freeing transaction 59705: 0x20073558
DBG: ../../../api/oc_buffer.c<process_thread_message_buffer_handler:193>: Outbound network event: unicast message
Outgoing message to: [192.168.0.111]:59264
```

Figure 5.5: HAN server POST response

5.5.2. Smart home underlying services

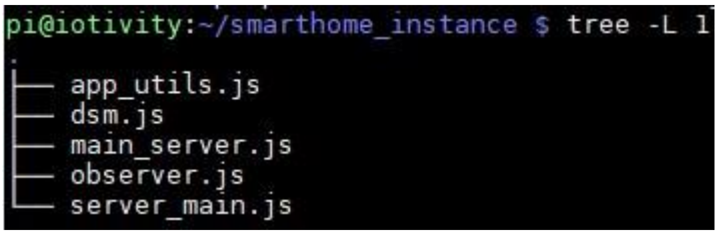
The daemon services started at boot time are the IoT REST API server and the main server service, the appliance observation service, and the DRM service.



```
{
  "name": "iotsmarthome",
  "version": "1.0.0",
  "description": "master project iot server",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "check-internet-connected": "^2.0.1",
    "iot-rest-api-server": "^0.5.0",
    "magic-globals": "^0.9.1",
    "mysql": "^2.17.1",
    "parse": "^2.6.0",
    "request": "^2.88.0",
    "underscore": "^1.9.1",
    "inquirer": "^7.0.0"
  }
}
```

Figure 5.6: JavaScript packages used for services development

The services are developed using Node JS with popular JavaScript libraries as shown in figure 5.6 above. The source for these services is developed within 5 scripts (see figure 5.7) and can be seen in Appendix E-F.



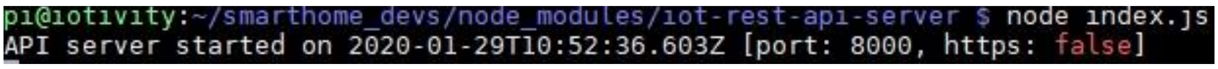
```
pi@iotivity:~/smarthome_instance $ tree -L 1
.
├── app_utils.js
├── dsm.js
├── main_server.js
├── observer.js
└── server_main.js
```

Figure 5.7: Smart home services scripts

5.5.2.1. IoT REST service

The gateway service uses the JavaScript IoTivity package named the IoT-restAPI-server. Version 0.5.0 for backward compatibility is used. In the figure below, is the logs from the IoT REST services that started the HTTP server on the localhost (127.0.0.1) (raspberry pi) on port 8000. This service is started by running a JS file (index.js) as shown in figure 5.8 below.

.

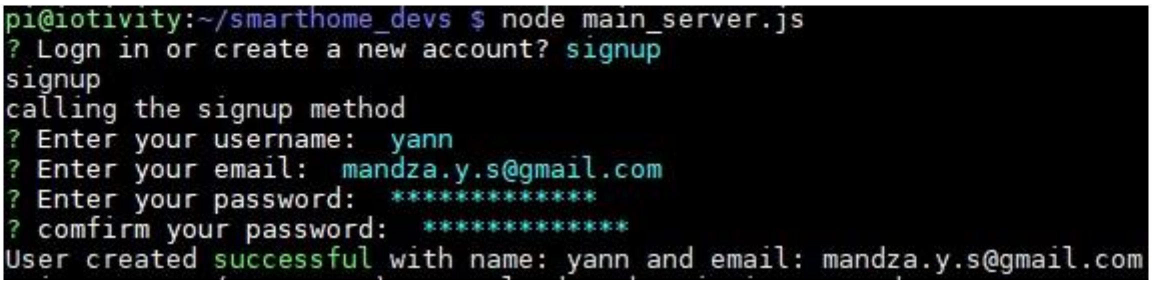


```
pi@iotivity:~/smarthome_devs/node_modules/iot-rest-api-server $ node index.js
API server started on 2020-01-29T10:52:36.603Z [port: 8000, https: false]
```

Figure 5.8: IoT REST server logs

5.5.2.2. Parse gateway service

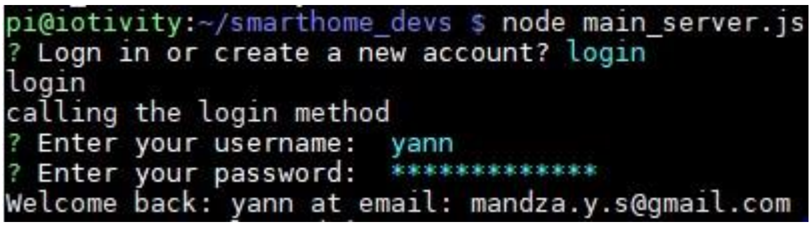
The Parse gateway service is the boot entry for the gateway services that handle connectivity to the online parse cloud on Back4App, launching of the resource observing, and the DRM daemon services. The service starts with an authentication procedure to either login/signup for a new installation of the system (new smart home system). This a basic securing feature to certify the smart home user.



```
pi@iotivity:~/smarthome_devs $ node main_server.js
? Logn in or create a new account? signup
signup
calling the signup method
? Enter your username:  yann
? Enter your email:  mandza.y.s@gmail.com
? Enter your password:  *************
? comfirm your password:  *************
User created successful with name: yann and email: mandza.y.s@gmail.com
```

Figure 5.9: Sign up authentication

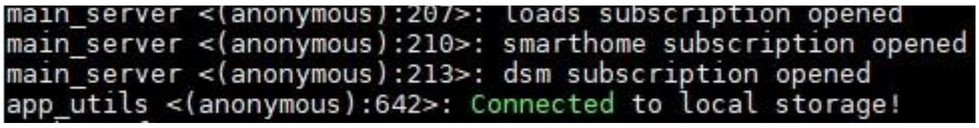
The Gateway service (main_server.js) starts by detecting a previous installation (a registered parse user). It thus asks the user to sign up (figure 5.9 above) for a first-time installation or to log in to authenticate the smart home (figure 5.10 below).



```
pi@iotivity:~/smarthome_devs $ node main_server.js
? Logn in or create a new account? login
login
calling the login method
? Enter your username:  yann
? Enter your password:  *************
Welcome back: yann at email: mandza.y.s@gmail.com
```

Figure 5.10: Login authentication

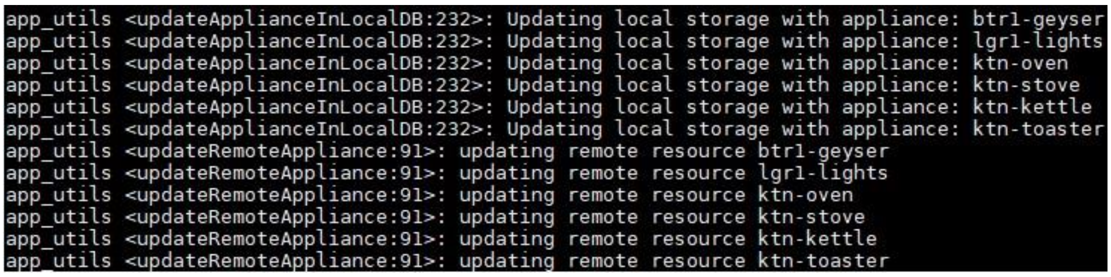
After authentication, the server starts a subscription (Live Query clients) to the Parse clouds on Back4App to listen to query updates on the loads, DRM, and the related smart-home objects (figure 5.11). Following these steps, the service connects to the local storage (MySQL database storing appliances properties locally) as well the remote parse "Loads" object. This is also a synchronization step to ensure both the local and remote storage are homogenous.



```
main_server <(anonymous):207>: loads subscription opened
main_server <(anonymous):210>: smarthome subscription opened
main_server <(anonymous):213>: dsm subscription opened
app_utils <(anonymous):642>: Connected to local storage!
```

Figure 5.11: Live Query subscriptions

Next, the service issue resources discovery and retrieves each resource property and thus update both the local MySQL storage and the remote Parse corresponding objects as shown in figure 5.12.

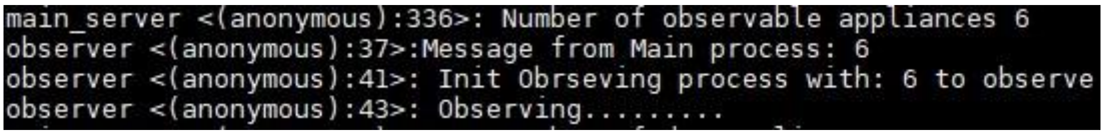


```
app_utils <updateApplianceInLocalDB:232>: Updating local storage with appliance: btr1-geyser
app_utils <updateApplianceInLocalDB:232>: Updating local storage with appliance: lgr1-lights
app_utils <updateApplianceInLocalDB:232>: Updating local storage with appliance: ktn-oven
app_utils <updateApplianceInLocalDB:232>: Updating local storage with appliance: ktn-stove
app_utils <updateApplianceInLocalDB:232>: Updating local storage with appliance: ktn-kettle
app_utils <updateApplianceInLocalDB:232>: Updating local storage with appliance: ktn-toaster
app_utils <updateRemoteAppliance:91>: updating remote resource btr1-geyser
app_utils <updateRemoteAppliance:91>: updating remote resource lgr1-lights
app_utils <updateRemoteAppliance:91>: updating remote resource ktn-oven
app_utils <updateRemoteAppliance:91>: updating remote resource ktn-stove
app_utils <updateRemoteAppliance:91>: updating remote resource ktn-kettle
app_utils <updateRemoteAppliance:91>: updating remote resource ktn-toaster
```

Figure 5.12: Resources storage updates

5.5.2.3. Observing service

Following the initialization step, the main service starts a child process to observe for updates on the local appliances and push those to parse server on Back4App. This process listens to changes on 6 appliances (figure 5.13 below).

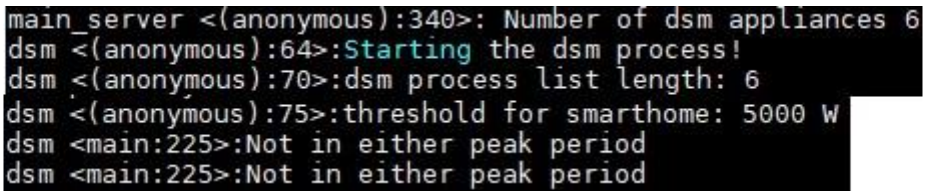


```
main_server <(anonymous):336>: Number of observable appliances 6
observer <(anonymous):37>:Message from Main process: 6
observer <(anonymous):41>: Init Obrseving process with: 6 to observe
observer <(anonymous):43>: Observing........
```

Figure 5.13: Observing service logs

5.5.2.4. DRM service

The DRM service handles the last scenario of the case study, it implements the algorithm for the 5 KW peak shaving scheme. The service is also started as a child service from the main service based on activation from the user registered via live query request on the DRM parse object. As shown in figure 5.14, the DRM handles a list of 6 appliances as defined in table 5.4-1. The logs here, show that the system is outside a peak period.

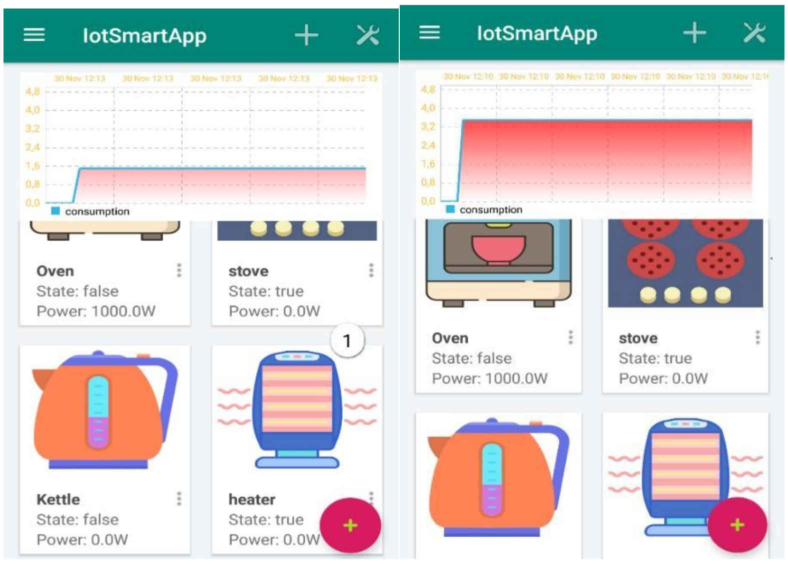


```
main_server <(anonymous):340>: Number of dsm appliances 6
dsm <(anonymous):64>:Starting the dsm process!
dsm <(anonymous):70>:dsm process list length: 6
dsm <(anonymous):75>:threshold for smarthome: 5000 W
dsm <main:225>:Not in either peak period
dsm <main:225>:Not in either peak period
```

Figure 5.14: DRM service logs

5.5.3. Feedback via Energy App

Regarding the Energy monitoring scenario, the platform's ability to provide space agnostic, real-time feedback is under evaluation.



Oven
State: false
Power: 1000.0W

stove
State: true
Power: 0.0W

Kettle
State: false
Power: 0.0W

heater
State: true
Power: 0.0W

Oven
State: false
Power: 1000.0W

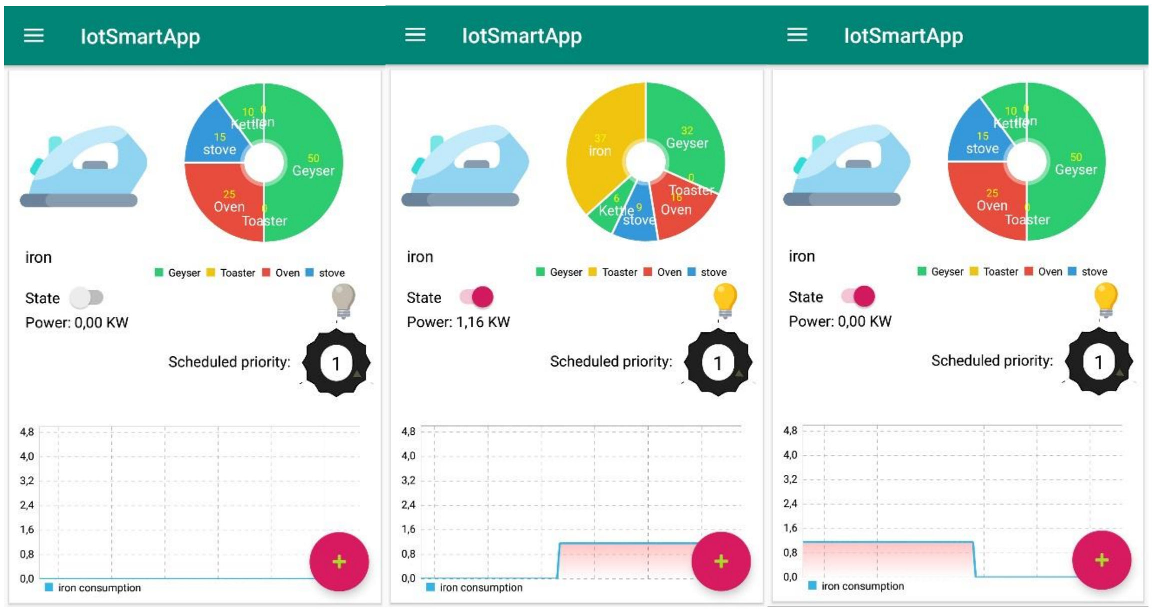
stove
State: true
Power: 0.0W

Figure 5.15: Energy Monitoring on IotSmartApp

Using a Sony Xperia Z5 smartphone, the Energy feedback was tested on the platform using the IotSmartApp as shown in figure 5.15 above. The energy consumption is presented in engaging visual tools both graphic and textual with compelling colours (red under the consumption curve). The evaluation shows that feedback can be dispatch via the platform within ~3 seconds from HAN to the Back4App clouds and the smartphone App.

5.5.4. Home automation via Energy App

As for home automation, the platform ability to provide space agnostic on/off control of the home appliance is evaluated. This Evolution is based on the setup of figure 5.17 in which two appliances (kettle and Iron) are used to demonstrate the system time and space agnostic on/off control via the platform.



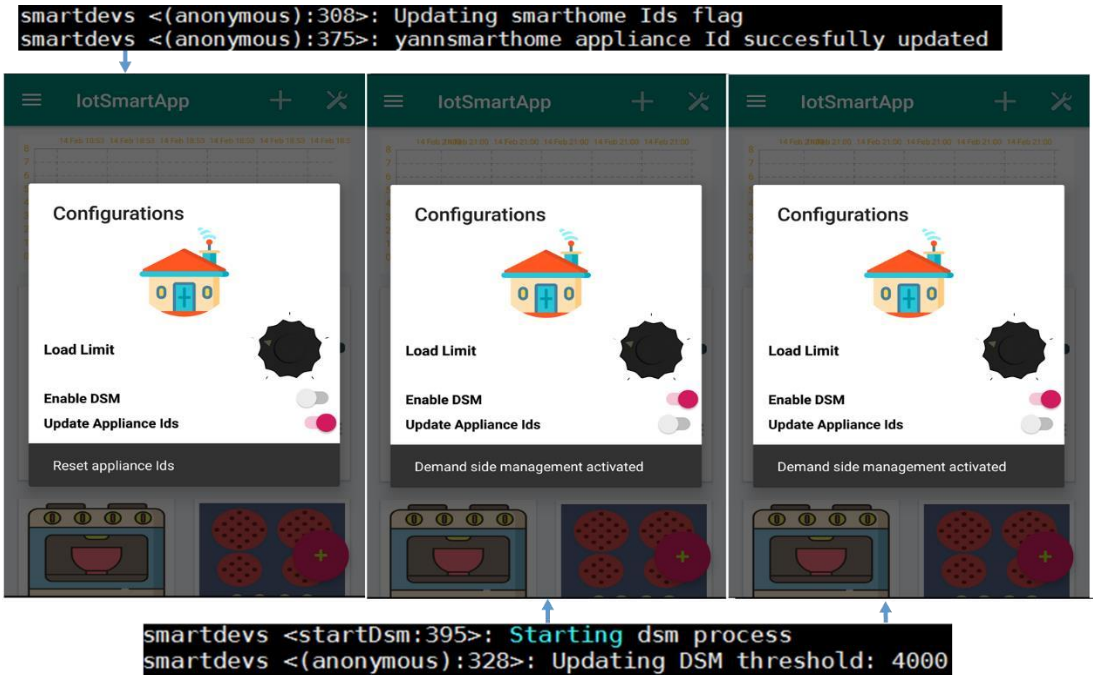
(a) (b) (c)

Figure 5.16: Home automation with IotSmartApp; (a) appliance is turn off; (b) appliance is turn off; (c) appliance power consumption off

In figure 5.16 is presented feedback about home automation via the IotSmartApp. This experimentation targets an iron-rated 1200W within the tested setup. On (a) the iron is off, thus its state is false (the lamp is grey). In this iron, consumptions share no part in the total house consumption as seen on the pie-chart. On (b) the iron is turn on (lamp is yellow), the consumption (power) at that instant was recorded as 1.16 KW which is also a graph on the line chart below on the App screen. This is practically demonstrated in figure 5.3, where the Arduino server connected to the physical appliance control circuit is activated (red light is on, kettle light is on for that instance). The pie chart depicting the total consumption then registers the iron share at 34%. On (C) the iron internal workings have turn power consumption off (KW) though the user still has the appliance off (lamp off). This can is emphasized on the graph that shows the iron consumptions falling to zero. This behaviour graphically represented enhance the feedback and help user to understand the working of their appliances. Finally, the interaction (on/off) from the user on IotSmartApp to appliance connected to IoTivity server on HAN within the platform takes about ~6s for bi-directional updates.

5.5.5. DRM via Energy App

The DRM scenario was tested within the platform using an Energy App developed for the purpose. In figure 5.17, configuration windows are proposed to the user to manage the DRM algorithm threshold (using the knob), reset the smart home's appliance IDs, and activate/de-activate (via the switch widgets) the DRM service running on the home server.



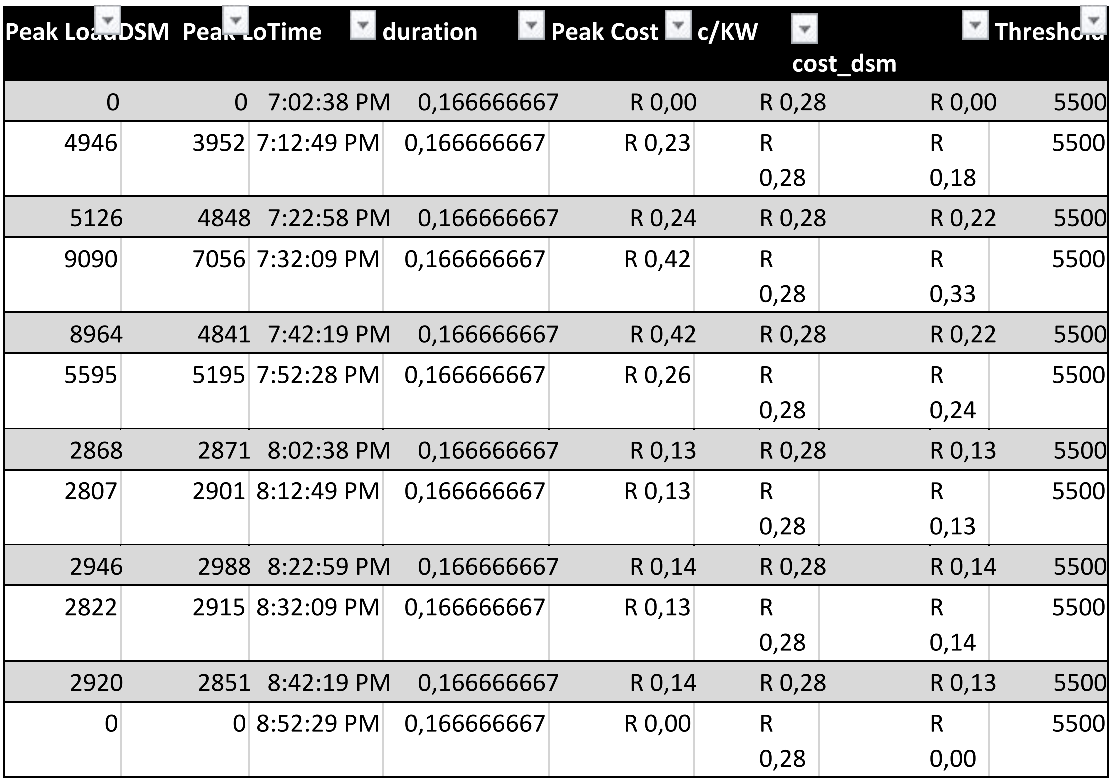
```
smartdevs <(anonymous):308>: Updating smarthome Ids flag
smartdevs <(anonymous):375>: yannsmarthome appliance Id succesfully updated
```

```
smartdevs <startDsm:395>: Starting dsm process
smartdevs <(anonymous):328>: Updating DSM threshold: 4000
```

Figure 5.17: DRM with Energy App

When the user activates the DRM service, both the new status and threshold are passed to the listening home server via the Live Query mechanism. The output of the algorithm for analysis was logged and plotted to appreciate the benefit of the peak-saving algorithm that was implemented. For this scenario, the maximum allowable peak demand to satisfy equation (4.10) is 5KW with a 10% positive margin (5.5KW. The experiment considered the morning peak running from 7 am to 9 am (3 hours) (which differs from the evening peak period by the difference of priority settings which are assumed as defined in table 5.1) and sample the consumption at 10 min duration. However, a timeframe of 5 min was used to simulate the peak period running the algorithm at 10 s then normalizing the results that stored during the simulation and presented in table 5.2 below.

Table 5.2: DRM Simulation results



Peak LoadDSM	Peak Lo	Time	duration	Peak Cost	c/KW	cost_dsm	Threshold
0	0	7:02:38 PM	0,166666667	R 0,00	R 0,28	R 0,00	5500
4946	3952	7:12:49 PM	0,166666667	R 0,23	R 0,28	R 0,18	5500
5126	4848	7:22:58 PM	0,166666667	R 0,24	R 0,28	R 0,22	5500
9090	7056	7:32:09 PM	0,166666667	R 0,42	R 0,28	R 0,33	5500
8964	4841	7:42:19 PM	0,166666667	R 0,42	R 0,28	R 0,22	5500
5595	5195	7:52:28 PM	0,166666667	R 0,26	R 0,28	R 0,24	5500
2868	2871	8:02:38 PM	0,166666667	R 0,13	R 0,28	R 0,13	5500
2807	2901	8:12:49 PM	0,166666667	R 0,13	R 0,28	R 0,13	5500
2946	2988	8:22:59 PM	0,166666667	R 0,14	R 0,28	R 0,14	5500
2822	2915	8:32:09 PM	0,166666667	R 0,13	R 0,28	R 0,14	5500
2920	2851	8:42:19 PM	0,166666667	R 0,14	R 0,28	R 0,13	5500
0	0	8:52:29 PM	0,166666667	R 0,00	R 0,28	R 0,00	5500

The data from Table 5.2 is used to plot the total peak load dynamics during and outside peak times, and the related consumption cost as shown in figure 5.18 below. The grey and blue curve of the figure denote the load profile with and without the demand management respectively whereas the brown and green curves represent the peak cost of consumption with and without demand management.

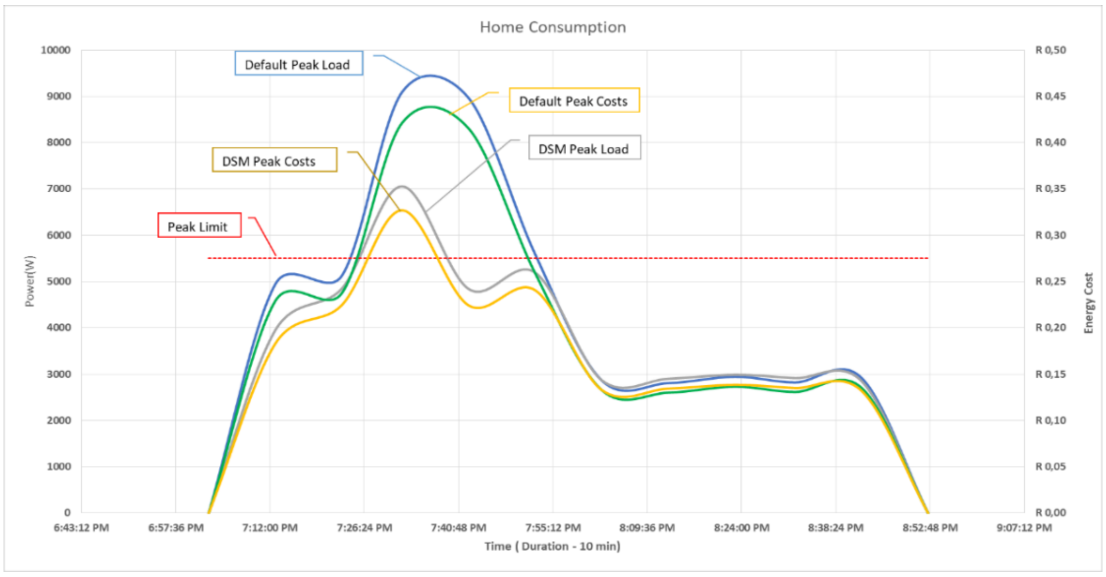


Figure 5.18: Peak load profiling through IoT platform

The Redline shows the maximum allowable demand threshold (about 5.5KW). When the demand exceeds the peak limit, the DRM service controls the appliances on the Table and turns some off according to the priority assigned. The DRM load profile (brown curve) peak is lowered, and the valleys are filled as expected of a peak shaving algorithm. The maximum peak is reduced from ~ 9KW (blue curve) on Default peak load to ~ 7KW on DRM profile this represents a ~ 17% reduction of peak load. However, the DRM peak overshoots the required threshold during the DRM operation. This is mainly because the DRM service computes the total appliance consumption every 10s couple to minor in code execution delays (internet latencies and underlying response to resource request). Thus, if the demand increases or changes rapidly within the sampling period, the DRM service gets the new demand with this delay and generates the controlling signal only after that delay. Hence the total load overshoots the peak limit. Nevertheless, the DRM load profile shows that the demand promptly falls back below the peak limit after performing the peak shaving algorithm.

5.6. Results discussion

The overall results support the platform goals of providing energy monitoring, home automation and DRM activities to customer locally or in remote location. However, the requirement of low-cost and miniaturization present noticeable performance issues in term of hardware memory constrained and response time. That is Arduino AVR presenting low RAM are not able to handle all essential requirements of the IoTivity stack (device and resource provisioning as well DTLS security). A 512K SRAM external memory shield compatible with the AVR memory bus was added. Although this solution stabilized heap allocations, the rather slow AVR CPU clock did little to increase its response time.

5.7. Chapter Summary

In this chapter was described the case study used to demonstrate the platform effectiveness and performance according to research objectives. After defining the requirements and objectives of each scenario, the results were presented as evaluated within the platform. Being dependant on underlying services both running on the home gateway, and on the cloud platform, this chapter mainly present the different tools used to developed and deployed the backbone services as well as the interaction between all services as they are run through the raspberry PI and the Back4App cloud. Energy feedback and HA via the Energy App running on Android (Sony Xperia Z5) was presented and discussed. Lastly, the DRM algorithm implemented for peak shaving scenario to demonstrate energy management through the platform was described. Then, were presented and discussed the governing equations and the different assumptions used to perform the simulation of appliance consumption. Then, were presented the DRM peak shaving algorithm results graphically highlighting its influence on consumption using the different tools proposed within the platform.

CHAPTER SIX
CONCLUSION AND RECOMMENDATIONS

6.1. Conclusion

This project was conceived to participate in the growing research regarding the modernization of the electric grid within the smart grid effort to better handle the growing peak demand and traditional grid limitations in the residential sector.

Therefore, “Cloud-based IoT platform for Energy management applications” an efficient and performant communication platform leveraging smart grid IoT enabling was presented in this thesis to provide smart energy management applications particularly in domestic places within the South African context. In this thesis was addressed the semantic gaps of IoT regarding interoperability, scalability as well as the cost and availability of technology issues as it pertains to HEMS. The thesis focused on the architectural design and backend requirements of an IoT platform around open source IoT technologies and developed a prototype full-stack system providing an experimental platform to perform smart-grid-related interventions in homes.

6.2. Meeting the research objectives

The objective defined for this thesis included:

1. Design and implement a responsive IoTivity-based HAN to handle IoT semantic gaps (devices interoperability), thereby increasing the miniaturizations of HAN devices and lowering cost,
2. Design IoTivity smart plugs for interfacing existing home appliances,
3. Optimize and scale the HAN using the cloud as BaaS to simplify the platform backend requirements,
4. Develop an Android-based Energy management App leveraging the cloud BaaS.

To reach these objectives, chapter 3 defined the specifications for the platform and selecting the technological tools required. In chapter 4 described the experimental setup developed to carry out the objectives. In chapter 5 was defined the characteristic, configurations, and requirements of each experimental scenario focusing on the DRM setup. Later in this chapter, the experimental results were presented as carried out through the platform. Thus, these objectives were addressed using the OCF IoTivity middleware which effectively provided interoperability of devices (Arduino, ESP32, Raspberry PI) and protocols (HTTP/S, CoAP), scalability as new motes could be plugged into the design without disturbing the current activities. Experimentation showed that WIFI and Ethernet devices could uniformly exchange data through the IoTivity HAN. The IoTivity smart plug were effectively design and deployed with sensing and actuating interfaces to any common household appliances (although focus was place on resistive ones). For higher MCU resources management and response time, two RTOS (Contiki RTOS and FreeRTOS) were used and adapted. Though FreeRTOS is inherent to ESP32, Contiki was chosen as being the basis for the IoTivity-Lite stack and lightweight for Arduino. The IoTivity firmware correctly and effectively enabled the motes to operate as HAN resources servers responding within latencies of ~3-5 second to CRUDN request from both local and remote resources clients. Cloud connectivity via the Back4App Parse BaaS clearly augment the performance and scalability of the platform. The Parse subscription mechanism greatly reduced computation and stack constraint allowing the local network architecture to focus solely on local resource requests. This has the benefit of increasing the system overall response time. Energy management, that consumption monitoring appliance operation control and DRM interventions was simplified in this work by providing a two-way communication between consumers and their residential load via a Mobile App. Through the engaging graphics and notifications supported by the Back4App Parse subscription and query mechanism, consumer can in real-time within a responsive interaction effectively engage their load.

6.3. Contributions of this research

The smart grid concept is trending amongst researchers, leading Energy Utilities to slowly deploy AMI technologies in the current constraining economic and technological conditions within developing context. However, the stress on the traditional grid and the yearly increasing residential load call for efficient energy sustainability alternatives able to take advantage of the current advances in technology while being scalable and interoperable to smart grid future upgrades and investment. To accomplish this vision, the internet and its related technology are regarded as suitable tools in the necessary transition from the traditional to the smart grid. Therefore, this thesis strives to participate in this transition to sustainable energy consumption by leveraging Internet dependant technology in IoT, cloud technologies, embedded design, mobile applications to provide a two-way Energy management platform that mitigate the complexities of existing HEMS, the performance of HAN, implementation cost favouring their increasing penetration in households through hardware miniaturization. Thus, this work contributes to the current government and utilities goals to bring energy consumption literacy, action tacking as well as management of household consumption at the appliance levels to all parties. these goals are made possible via a platform that incorporates the existing appliances and residential connectivity facilities while being interoperable, scalable, and cost-effective for higher penetration of the smart grid vision in the residential sector.

6.4. Recommendations

- Security can be increased in the platform using the IoTivity onboarding and provisioning mechanism to authenticate the client that interfaces to the HAN resource server. This capability was not fully implemented because of software inconsistency with the IoT-rest-API-server. Thus, security was mainly at the cloud interface isolating IoTivity LAN resource servers.

- IoTivity Cloud, OCF has updated its IoTivity-Lite framework to add a cloud interface to the IoTivity network. This facility can be used to remove the need for IoT-rest-API-server reducing the development load and facilitates maintenance.
- Higher-end embedded device for HAN servers able to handle multiple clients while maintaining a fast response time was observed an issue with AVR motes, and in some capacities with the DUE servers due its reduced processing speed and constrained memory. A miniaturized higher memory MCU running at faster clock would provide and smother response time.
- Providing complete Offline access to services on the platform. It is necessary to offer the user with complete offline experience of the platform in a developing context where internet connection may be intermittent. The has some limited preparation for such, but the Energy App should still be able to provide Energy services on the local network (users are at home).
- Wireless communication, here WIFI should be adapted to all HAN devices for easier penetration and adaptation in residential places. Technology with embedded wireless protocol should be used to optimize the HAN data communication. The ESP32 device was adopted with appropriate IoTivityLite network firmware modifications.
- Smart grid signals from the Energy utility can take advantage of this platform. but the interface needs to be fully defined from the cloud interface this can be a cloud Job that needs to monitor or listen via an API provided by Utility to smart grid incentives and propagating these to home gateways.

REFERENCES

Abdelsamea, M.H.A., Zorkany, M. and Abdelkader, N. (2016) “Real Time Operating Systems for the Internet of Things, Vision, Architecture and Research Directions,” *Proceedings - 2016 World Symposium on Computer Applications and Research, WSCAR 2016*, (September 2018), pp. 72–77. doi:10.1109/WSCAR.2016.21.

Abdulrahman, T.A. *et al.* (2016) “Design, Specification and Implementation of a Distributed Home Automation System,” *Procedia Computer Science*, 94(IoTNAT), pp. 473–478. doi:10.1016/j.procs.2016.08.073.

Ahmad, M.W. *et al.* (2016) “Building energy metering and environmental monitoring - A stateof-the-art review and directions for future research,” *Energy and Buildings*, 120, pp. 85–102. doi:10.1016/j.enbuild.2016.03.059.

Albu, M. and Heydt, G.T. (2003) “On the use of RMS values in power quality assessment,” *IEEE Transactions on Power Delivery*, 18(4). doi:10.1109/TPWRD.2003.817518.

Asare-Bediako, B. (2019) *SMART energy homes and the smart grid : a framework for intelligent energy management systems for residential customers*. doi:10.6100/IR781632.

Baccelli, E. *et al.* (2018) “RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT,” *IEEE Internet of Things Journal*, 5(6), pp. 4428–4440. doi:10.1109/JIOT.2018.2815038.

Beligianni, F. *et al.* (2016) “An internet of things architecture for preserving privacy of energy consumption,” *Mediterranean Conference on Power Generation, Transmission, Distribution and Energy Conversion (MedPower 2016)*, pp. 107 (7 .)-107 (7 .). doi:10.1049/cp.2016.1096.

Coval, P. and Sun, Z. (2017) “IoTivity: From Devices to the Cloud,” in *Free and Open Source Software Developers’ European Meeting*. Berlin: fosdem, pp. 1–28.

Dang, T.-B. *et al.* (2017) “On Evaluating IoTivity Cloud Platform,” *Computational Science and Its Applications -- ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part V*. Edited by O. Gervasi et al., 10408, pp. 137–147. doi:10.1007/978-3319-62404-4_10.

Department of Energy (2012) “A survey of energy-related behaviour and perceptions in South Africa: The residential sector,” 1(1), p. 118. doi:ISBN: 978-1-920435-04-2.

Dlodlo, N. *et al.* (2015) “Research trends in existing technologies that are building blocks to the internet of things,” *Lecture Notes in Electrical Engineering*, 313, pp. 539–548. doi:10.1007/978-3-319-06773-5_72.

Dwivedi, A.K., Tiwari, M.K. and Vyas, O.P. (2009) “Operating Systems for Tiny Networked Sensors : A Survey,” *International Journal of Recent Trends in Engineering*, 1(2), pp. 152–157.

Elfström, K. (2017) *Evaluation of IoTivity: A Middleware Architecture for the Internet of Things*. KTH ROYAL INSTITUTE OF TECHNOLOGY.

Gyrard, A. and Serrano, M. (2016) “Connected smart cities: Interoperability with SEG 3.0 for the internet of things,” *Proceedings - IEEE 30th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2016*, (2), pp. 796–802. doi:10.1109/WAINA.2016.151.

Haider, H.T., See, O.H. and Elmenreich, W. (2016) “A review of residential demand response of smart grid,” *Renewable and Sustainable Energy Reviews*, 59, pp. 166–178. doi:10.1016/j.rser.2016.01.016.

Human Sciences Research Council (2013) *A survey of energy related behaviour and perceptions in South Africa: the residential sector*, *Report Compiled for the Department of Energy of South Africa*.

Hussain, H.M. *et al.* (2018) “An efficient demand side management system with a new optimized home energy management controller in smart grid,” *Energies*, 11(1), pp. 1–28. doi:10.3390/en11010190.

Jerabandi, M. and M Kodabag, i M. (2017) “Internet of Things Based Technology for Smart Home System : A Generic Framework,” *International Journal on Recent and Innovation Trends in Computing and Communication* [Preprint].

Jian, M.-S. *et al.* (2017) “IOT base smart home appliances by using Cloud Intelligent Tetris Switch,” *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 589–592. doi:10.23919/ICACT.2017.7890158.

Kailas, A., Cecchi, V. and Mukherjee, A. (2012) “A Survey of Communications and Networking Technologies for Energy Management in Buildings and Home Automation,” 2012. doi:10.1155/2012/932181.

Kalmeshwar, M. and K S, Assoc.P.Dr.N.P. (2017) “Internet Of Things: Architecture,Issues and Applications,” *International Journal of Engineering Research and Applications*, 07(06), pp. 85–88. doi:10.9790/9622-0706048588.

Kang, B. and Choo, H. (2017) “An experimental study of a reliable IoT gateway,” *ICT Express* [Preprint]. doi:10.1016/j.icte.2017.04.002.

Kavyashree, E. (2018) “6LoWPAN NETWORK USING CONTIKI OPERATING SYSTEM,” *Researchgate.Net*, (June), pp. 300–310. doi:10.21467/proceedings.1.48.

Khan, A. *et al.* (2019) “A priority-induced demand side management system to mitigate rebound peaks using multiple knapsack,” *Journal of Ambient Intelligence and Humanized Computing*, 10(4), pp. 1655–1678. doi:10.1007/s12652-018-0761-z.
Khatu, M. *et al.* (2015) “Implementation of Internet of Things for Home Automation,” *International Journal of Emerging Engineering Research and Technology*, 3(2), pp. 7–11.
Larsson, A. and Nimmermark, E. (2016) *Comparison of IoT frameworks for the smart home*.
Le, D.-T. (2017) “On Evaluating IoTivity Cloud Platform,” in *Lecture Notes in Computer Science*. doi:10.1007/978-3-319-62404-4. *Learn | OpenEnergyMonitor* (no date). Available at: https://learn.openenergymonitor.org/electricity-monitoring/ac-power-theory/arduino-maths (Accessed: November 8, 2021).
Lee, J.C., Jeon, J.H. and Kim, S.H. (2016) “Design and implementation of healthcare resource model on IoTivity platform,” *2016 International Conference on Information and Communication Technology Convergence, ICTC 2016*, pp. 887–891. doi:10.1109/ICTC.2016.7763322.
Lobaccaro, G., Carlucci, S. and Löfström, E. (2016) “A review of systems and technologies for smart homes and smart grids,” *Energies*, 9(5), pp. 1–33. doi:10.3390/en9050348. Longe, O.M. *et al.* (2017) “Consumer preference electricity usage plan for demand side management in the smart grid,” *SAIEE Africa Research Journal*, 108(4), pp. 174–183.
Macieira, T. (2016) “IoTivity : The Open Connectivity Foundation and the IoT Challenge,” in *Embedded Linux Conference*. Berlin.
Maloor, K. *et al.* (2015) “IoTivity Programmer ’ s Guide – Resource Encapsulation,” *TIZEN Development Summit*, pp. 0–95.
Maloor, K. (2017) “IoTivity-Constrained:IoT for tiny devices,” in *Open IoT Summit North*.
Maloor, K. (2019) “Kishen Maloor , Intel,” in *OCF EU Developer Training*. Budapest. Minoli, D., Sohraby, K. and Occhiogrosso, B. (2017) “IoT Considerations, Requirements, and Architectures for Smart Buildings – Energy Optimization and Next Generation Building Management Systems,” *IEEE Internet of Things Journal*, 4(1), pp. 1–1. doi:10.1109/JIOT.2017.2647881.
Miron-Alexe, V. (2016) “Comparative study regarding measurements of different AC current sensors,” *2016 International Symposium on Fundamentals of Electrical Engineering, ISFEE 2016* [Preprint], (June 2016). doi:10.1109/ISFEE.2016.7803152.
Paridah, M. t *et al.* (2016a) “Advanced Metering Infrastructure Based on Smart Meters in Smart Grid,” *Intech*, i(tourism), p. 13. doi:http://dx.doi.org/10.5772/57353.
Paridah, M. t *et al.* (2016b) “We are IntechOpen , the world ’ s leading publisher of Open Access books Built by scientists , for scientists TOP 1 %,” *Intech*, i(tourism), p. 13. doi:http://dx.doi.org/10.5772/57353.
Perera, C. *et al.* (2014) “MOSDEN: An internet of things middleware for resource constrained mobile devices,” *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 1053–1062. doi:10.1109/HICSS.2014.137.
Piyare, R. (2013) “Internet of Things : Ubiquitous Home Control and Monitoring System using Android based Smart Phone,” *international Journal of Internet of Things*, 2(1), pp. 5–11. doi:10.5923/j.ijit.20130201.02.
Qayyum, F.A. *et al.* (2015) “Appliance Scheduling Optimization in Smart Home Networks,” *Access, IEEE*, 3, pp. 2176–2190. doi:10.1109/ACCESS.2015.2496117.
Ray, P.P. (2017) “A Survey of IoT Cloud Platforms,” *Future Computing and Informatics Journal*, 1(1–2), pp. 35–46. doi:10.1016/j.fcij.2017.02.001.
Razzaque, M.A. *et al.* (2016) “Middleware for internet of things: A survey,” *IEEE Internet of Things Journal*, 3(1), pp. 70–95. doi:10.1109/JIOT.2015.2498900.
Risteska Stojkoska, B.L. and Trivodaliev, K. V. (2017a) “A review of Internet of Things for smart home: Challenges and solutions,” *Journal of Cleaner Production*, 140(January), pp. 1454–1464. doi:10.1016/j.jclepro.2016.10.006.
Risteska Stojkoska, B.L. and Trivodaliev, K. V. (2017b) “A review of Internet of Things for smart home: Challenges and solutions,” (January). doi:10.1016/j.jclepro.2016.10.006.
Roussel, K., Song, Y.Q. and Zendra, O. (2015) “RIOT OS paves the way for implementation

of high-performance MAC protocols," *SENSORNETS 2015 - 4th International Conference on Sensor Networks, Proceedings*, pp. 5–14.

Sahadevan, A. *et al.* (2017) "An Offline Online Strategy for IoT Using MQTT," *Proceedings - 4th IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2017 and 3rd IEEE International Conference of Scalable and Smart Cloud, SSC 2017*, pp. 369–373. doi:10.1109/CSCloud.2017.34.

Serov, A.N., Serov, N.A. and Makarychev, P.K. (2019) "Evaluation of the Effect of Nonlinearity of the Successive Approximation ADC to the Measurement Error of RMS," in *2018 International Symposium on Industrial Electronics, INDEL 2018 - Proceedings*. doi:10.1109/INDEL.2018.8637630.

Sethi, P. and Sarangi, S.R. (2017) "Internet of Things : Architectures , Protocols , and Applications," 2017.

Soto, V.E.A. (2017) *PERFORMANCE EVALUATION OF SCALABLE AND DISTRIBUTED IOT PLATFORMS FOR SMART REGIONS*. Luleå University of Technology.

South Africa Connect (2013) *Executive Summary National policy and constitutional context Challenges of broadband*.

South African Department of Energy (2013) "Draft 2012 Integrated Energy Planning Report - Annexure A - Technical Report - Part 1 : Demand Modelling Report."

South African Department of Energy (2016) *Post-2015 National Energy Efficiency Strategy: Comments invited*.

Sun, Z. (2017) "GENIVI + OCF Cooperation," in *Samsung Open Source Group*, pp. 1–35.

Sutisna *et al.* (2019) "Power analyzer based arduino-uno validation using Kyoritsu KEW 6315 and Hioki 328-20," *IOP Conference Series: Materials Science and Engineering*, 550, p. 012024. doi:10.1088/1757-899x/550/1/012024.

Szablya., L. (2012) "Contributed Article Offered as an Exclusive to EnergyPulse Home Energy Management with or without an Advanced Metering Infrastructure [Revised Draft • 3322 Words]," *Electric Energy T&D Magazine*, October, p. 14.

Vinh, T. Le *et al.* (2015) "Middleware to integrate mobile devices, sensors and cloud computing," *Procedia Computer Science*, 52(1), pp. 234–243. doi:10.1016/j.procs.2015.05.061.

Zhang, Y. *et al.* (2015) "A Novel Multiobjective Optimization Algorithm for Home Energy Management System in Smart Grid," *Hindawi*, 2015, pp. 1–19.

APPENDIXES

Appendix A. Motes DAQ Modules Circuit Diagram

Below is the circuit diagram for the AVR and ARM Arduino. The circuit is divided in three sections mainly, the *control section* (with relay interface), the *power and communication,* and the *CT sensor interface section*

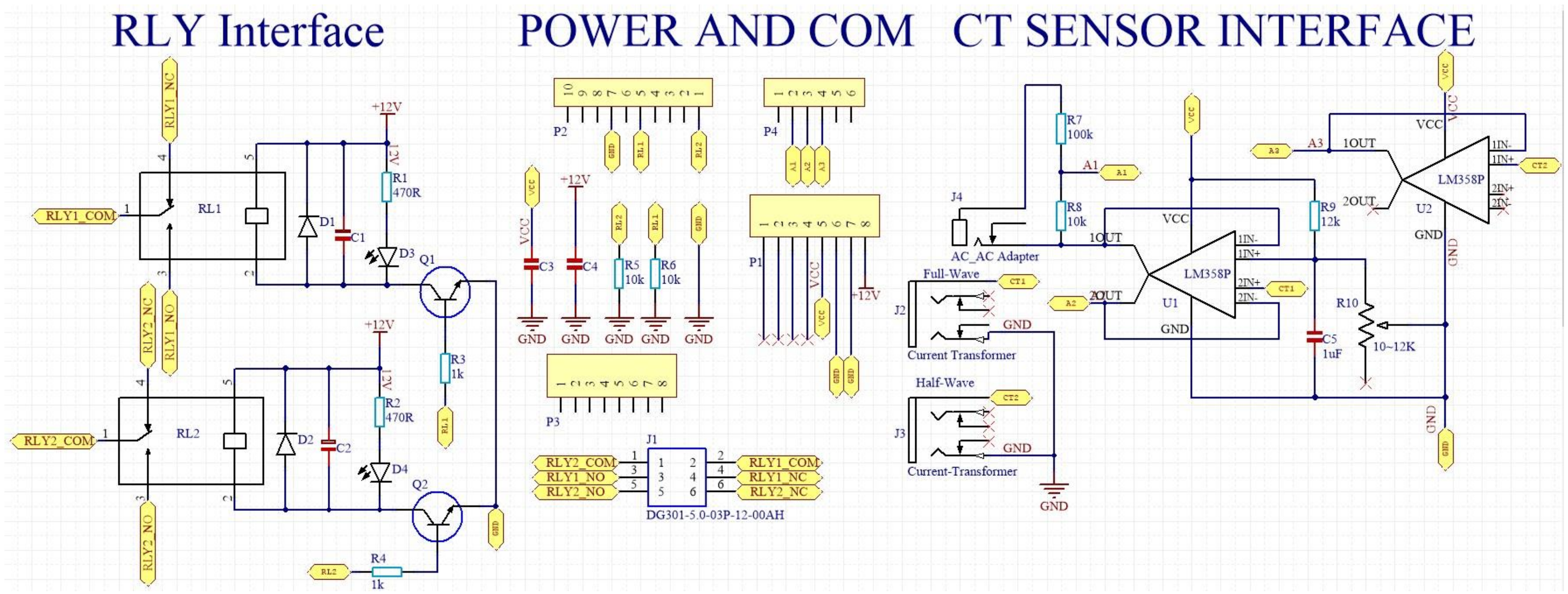


Figure A.1: Complete DAQ module for Arduino HAN resource servers

Below is the circuit diagram for the ESP32 based HAN resource server which is composed of similar section as that of the Arduino based DAQ modules.

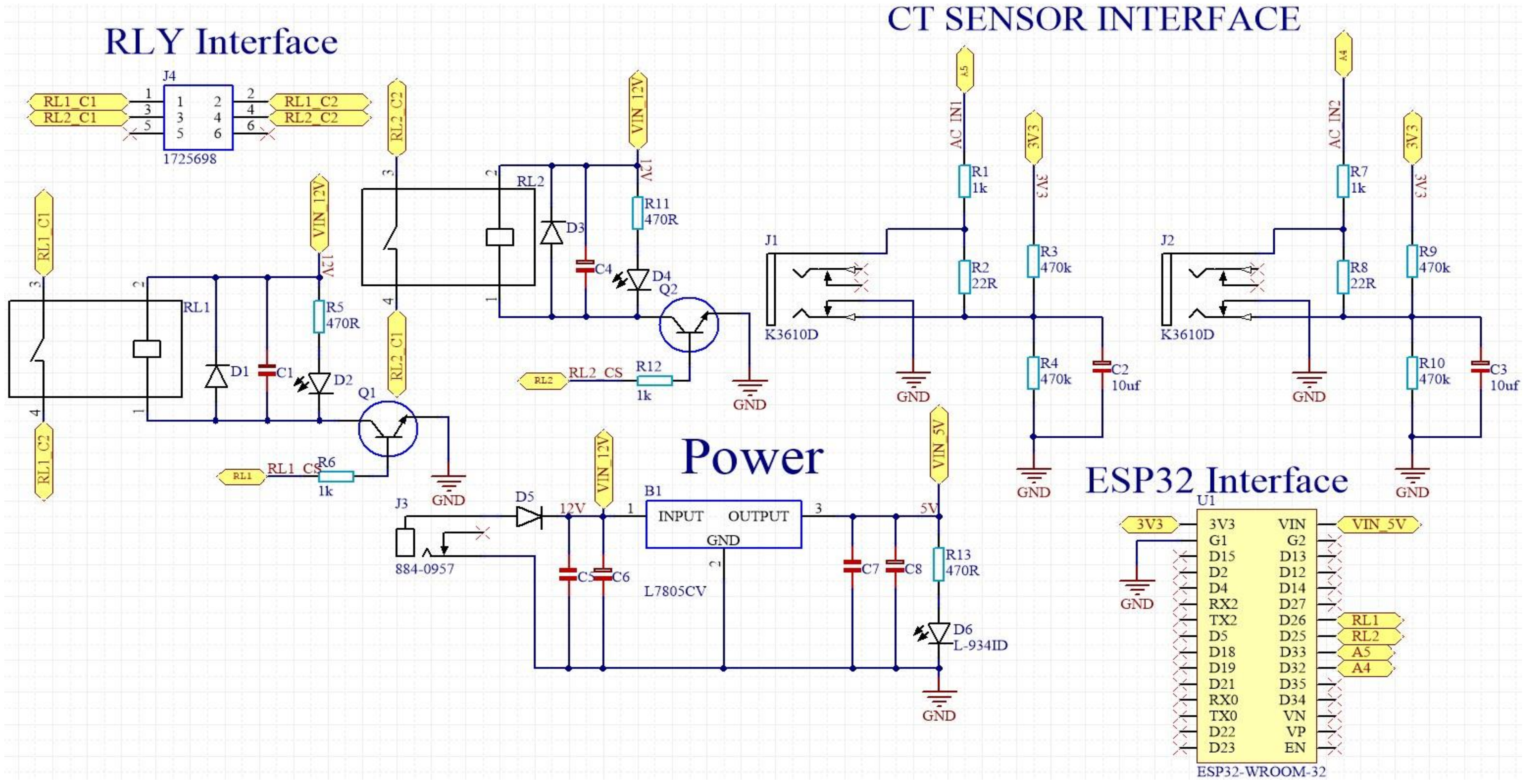
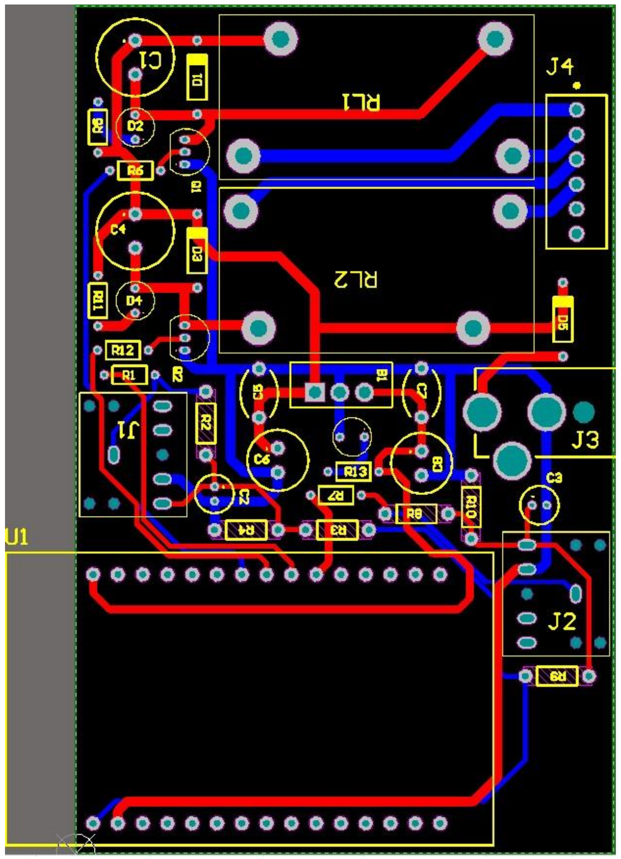


Figure A.2: Complete DAQ Module for ESP32 HAN resource servers

Appendix B. DAQ Modules PCB layout

Altium Designer 2017 was used to design and Manufacture the PCB for the ESP32 and Arduino DAQ, the final designed files are show in Figure B.1 below.


(a)

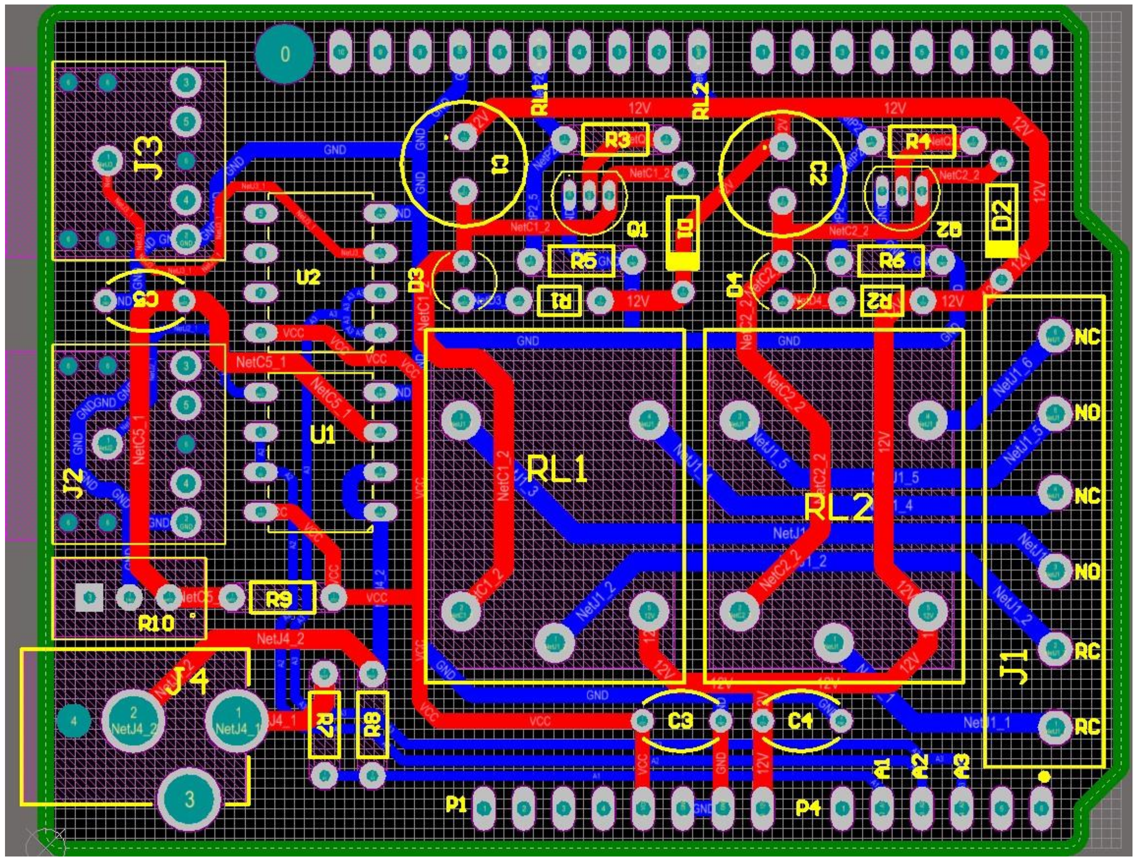

(b)

Figure B.1: PCB layout for HAN Resource Server DAQ module

(a) ESP32 interfacing PCB; (b) Arduino (AVR&ARM) DAQ module

Algorithm 1: Sensing circuit Interfacing and Power properties Calculation

```
1. initialized n = 0; inst_current = 0 ; inst_voltage =0
2. initialized dc_offset= 512(10bits) or 2048(12 bits)
3. initialized struct power_data = {0,0,0,0};
4. initialized sampleI = 0; sampleV = 0, lastSampleV = 0, phaseShiftedV = 0;
5. Initialized Vsens = 14.6V(~ 20% 12 V), Vmainsdef = 230 V, Vgain = 11
/*Arduino AVR*/
6. initialized Vref = readVcc(), ADC_counts = 1024;
/*ESP32 and Arduino ARM*/
7. initialized Vcc = 3300, ADC_counts = 4098;
8. intiailized sampling_window: 25 cycles at 50Hz or 500ms
9. initialized init_time = millis() or  esp_timer_get_time()
10. intialized sum_inst_power =0; sum_Irms = 0, sum_Vrms = 0;
/*ESP32*/
11. current_time = esp_timer_get_time()
/*Arduino*/
12. current_time =  millis()
13. while current_time  - init_time < sampling_window
    /*For phase angle adjustments*/
    14. lastSampleV = sampleV;
    /*ESP32*/
    15. inst_current = (adc1_get_raw((adc1_channel_t)appliance)  - dc_offset) * 30A/1V
    /*Arduino*/
    16. inst_current = (analogRead ( appliance )  -  dc_offset) * 30A/1V
    17. sampleV = ((analogRead ( A1 ) - dc_offset) * (Vref / ADC_counts))
    /*Voltage phase angle calibration*/
    18. phaseShiftedV = lastSampleV + PHASECAL * (sampleV - lastSampleV);
    19. inst_voltage = (phaseShiftedV * ((Vmainsdef * Vgain)/Vsens));
    20. power_data.inst_power = inst_voltage * inst_current; /*(equation
    21. sum_inst_power += power_data.inst_power;
    22. sum_Irms += sq(inst_current);
    23. sum_Vrms += sq(inst_voltage);
    24. n = n +1
25. end loop
26. power_data.Irms= sqrt(sum_Irms / n);
27. power_data.Vrms = sqrt(sum_Vrms / n);
28. power_data.Real_power = sum_inst_power / n;
29. power_data.Apparent_power = power_data.Irms * power_data.Vrms;
```

Figure C.1: Algorithm for appliance power properties computation

Algorithm 1: DRM Scenario algorithm

```
initialized Ppeak = [], PpeakCost = [];
initialized peakCapacity = 5000; // 5KW
initialized activeLoads = [], recycleLoads = [];
initialized timeStep = 15;   // the DSM algorithm is executed every 15 minutes
initialized peakTime = false;
initialized maxPeak = 0, avgPeak = 0;
initialized maxPeakCost = 0, avgPeakCost = 0;
initialized Ecost = 278.46 // c/kWh;
initialized lowPriorityLoad = {};
/*The main process main_server.js executed the DSM algorithm within as child
  process started with a msg data after an update event is recieve via Live Query*/
for(let i = 0; i < msg.loads.length; i++)
 loads.push(msg.loads[i])
end
// update threshold
peakCapacity = msg.threshold;
if peakTime then
    demand =  await computeDemand();
    Ppeak.push(demand);
    // compute and store energy cost
    var  energyPrice = (demand * 15 / 60 * 1000) * Ecost;
    PpeakPrice.push(energyPrice);
    if recycleLoad.length > 0
       // recycle low priority loads
       while demand < peakCapacity
       // get the lowest priority load and turn it on
       setLoadPower(getRecycleLoad())
       // update demand
       demand = await computeDemand();
      end
    end
    while(demand > peakCapacity)
       // get the lowest priority load in the loads list and turn it off
       lowPriorityLoad =  getLowPriorityLoad();
       resetLoads(lowLoad);
       demand = await computeDemand();
    end
    if lowPriorityLoad
       recycleLoads.push(lowPriorityLoad);
    end
else
    // calculate the average peak power and return the maximum peak load
    var sumPeak, sumCost, n;
    var maxPower = Ppeak[0];
    var maxCost = PpeakCost[0];
    for (let i = 0; i < Ppeak.length; i++)
       if Ppeak[i] > maxPower
          maxPower = Ppeak[i];
       end
       if PpeakCost[i] > maxCost
          maxCost = PpeakCost[i];
       end
       sumCost += PpeakCost[i]
       sumPeak += Ppeak[i];
    end
    avgPeakCost = sumCost / n;
    avgPeak = sumPeak / n;
    maxPeak = maxPower;
    maxPeakCost = maxCost;
end
```

Figure C.2: Algorithm for DRM simulation scenario

Appendix D. DSM Peak Shaving source code

```
var apputils = require("./app_utils");
var request = require('request');
const fs = require('fs') ; var
peakCapacity = 5500 ; // 5 KW var
loadsRatings =
[
  {name:'kettle', upper:3000000, lower:1200000},
  {name:'geyser', upper:3000000, lower:2800000},
  {name:'toaster',upper:1800000, lower:800000},
  {name:'oven', upper:5000000, lower:3000000},
  {name:'stove', upper:3000000, lower:1200000},
  {name:'iron', upper:1200000, lower:1000000}
];  var Ecost = 278.46 // c/KWH var averagePeakDemand = 0.0;
//KW var monringPeakDuration = 10800 ; // 3 hours from 7am to
10am var eveningPeakDuration = 7200 ;  // 2 hours from 6pm to
8pm
 var timeStep = 6000 // 50s
min var loadCount = 6;
  var peakActive =  false; var eveningPeakActive =  false; var
morningPeakActive =  false; var peakType = false; // 0:
morning; 1: evening; 2: not in peak var  Ppeak = []; var
PpeakCost = [];
var currentHours = 0;                                    // time value
telling us about the current hours of the peak period var
morningPeakHourBegin = 7;  // peak period start time : 7am
var morningPeakHourEnd = 9;            // peak period start time : 7am
var eveningPeakHourBegin = 18;         // peakperioad end time: 8pm or 18
hours
var eveningPeakHourEnd = 21;           // peakperioad end time: 8pm or 18
hours var run = false;
var loads = []; var
activeLoads = []; var
homeDemand = []; var
recycleLoads = []; var
n = 5; var stream =
null;


process.on('message', (msg) => {

  if(msg != null) {     var
appliances = msg.loads;
     if(appliances){
                console.log(`%s <%s:%d>:Starting the dsm process!`,__file,
__func, __line);
             for(let i = 0; i < appliances.length; i++){
     loads.push(appliances[i]);
            }                   console.log(`%s <%s:%d>:dsm
process list length:
${loads.length}`,__file, __func, __line);

  console.log(    '%s <%s:%d>: dsm loads '
__line,                                                                          ,__file
appliances);                                                                     ,
                                                                                 __func,
        }
  if(msg.threshold) {      peakCapacity = msg.threshold;
console.log(`%s <%s:%d>:threshold for smarthome: ${peakCapacity}
W`,  file,   func,   line);
  }
  var appliance = msg.appliance;   if(appliance) {
var appInfo = apputils.find(appliance, loads);
if(appInfo.state){      loads[appInfo.pos].priority =
appliance.priority;
                  %s <%s:%d>:updating priorityt
console.log(`
${appliance.name}:${appliance.priority}`,__file, __func, __line);
   }
      if(!appliance.state) {     console.log(`%s <%s:%d>:appliance
state: ${appliance.state}`,__file,
__func, __line);
if(recycleLoads.length > 0){
                  %s      <%s:%d>:load      to ,__file, __func, __line,
console.log(`     recycler`
recycleLoads);
    var appInfo = apputils.find(appliance, recycleLoads);
if(appInfo.state) {
     console.log(`   %s <%s:%d>:size of recycler list:
${recycleLoads.length}`,__file, __func, __line);
 recycleLoads = remove(recycleLoads[appInfo.pos], recycleLoads);
 console.log(`   %s <%s:%d>:size of recycler list:
${recycleLoads.length}`  ,__file, __func, __line);
                                        }
                                }
                        }
        }
  }
});

function getProperties(url, load) {
    var options = {
        url: url,
        method: 'GET',
        headers: {
            'User-Agent': 'request',
            'Content-Type': 'application/json'
        }
    };      return new Promise(function(resolve, reject)
{          request.get(options, function(err, resp, body)
{              if (err) {                     reject(err);
} else {

                var location = properties.name.split('
 )[0].toLowerCase();
load.location = location
                properties = JSON.parse(body)

'
```

```
                load.state = properties.state
load.current = properties.current
load.power = properties.power
resolve(load);
            }
        })
    })
}
///////////////////////////////////////////////////////////////////////////
var getDemand = async function() {   try    {     homeDemand = [];
for(let i = 0; i < loads.length; i++){        var load = {       location:
loads[i].location,         locationId:  loads[i].locationId,
applianceId: loads[i].applianceId,           name:        loads[i].name,
         deviceId:    loads[i].deviceId,           state:
loads[i].state,      current:     loads[i].current,             power:
loads[i].power
     };         load_url='http://localhost:8000/api/oic/'+
load.locationId + '/' + load.name +'?di='+ load.deviceId     try {
var data = await getProperties(load_url, load)
homeDemand.push(data.power);
                             %s <%s:%d>:Load power
console.info(`
${data.name}:${data.power}`,__file, __func, __line);
var loadInfo = apputils.find(data, loads);
     if(loadInfo.state) {      loads[loadInfo.pos].state =
data.state;        loads[loadInfo.pos].current =
data.current;     loads[loadInfo.pos].power = data.power;
for(let j = 0; j < loadsRatings.length; j++){      var
load = loads[loadInfo.pos];      var ratedLoad =
loadsRatings[j]      if(load.name == ratedLoad.name){
if(load.state) {             var appInfo =
apputils.find(load, activeLoads);
if(!appInfo.state) {
              activeLoads.push(load);
           } else {
activeLoads[appInfo.pos].state = load.state;
           }
                               %s <%s:%d>: Load ${loads[loadInfo.pos].name} is
console.log(`
on`,__file, __func, __line);
       } else {
         loads[loadInfo.pos].state = false;
                                %s <%s:%d>: Load ${loads[loadInfo.pos].name} is
console.log(`
off`,__file, __func, __line);
      }
      break;
     }
    }
   }
  } catch (err){
                             %s         <%s:%d>:,__file, __func, __line);
console.error(`              ${err}`
  }
 }         }
catch(err) {
```

```
                             %s          <%s:%d>: ,__file, __func, __line);
console.error(`     ${err}`
   }
}
// time step demand var computeDemand =
async function(){    await getDemand();
var demand = 0.0;    for(let i =0; i <
loads.length; i++){      var load =
loads[i];      console.error(`%s <%s:%d>:
Load state:
${load.name}:${load.state}` ,__file, __func, __line);
    demand += ((load.power / 1000.0) * load.state);
  }
  return demand;
}  var main = async function(){   if(!stream) {      stream =
fs.createWriteStream("append.txt", {flags:'a'});
  }   var date = new
Date();
console.log(`%s
<%s:%d>:${date.getHours()}:${date.getMinutes()}` ,__file, __func, __line);
  var currentHour = date.getHours();
  if(currentHour >= morningPeakHourBegin && currentHour <=
morningPeakHourEnd)  {              peakActive   =     true;
morningPeakActive = true;      peakType = 1;
                        %s <%s:%d>: Morning peak active` ,__file, __func, __line);
console.log(`
        } else if (currentHour >= eveningPeakHourBegin && currentHour <=
eveningPeakHourEnd)          {
peakActive     =          true;
eveningPeakActive    =    true;
peakType = 0;
                           %s     <%s:%d>:Evening     peak ,__file, __func,
console.log(`    active`
__line);
       } else {             peakActive
=   false;      morningPeakActive  =
false;    eveningPeakActive = false;
peakType = 2;
                        %s <%s:%d>:Not in either peak period` ,__file, __func,
console.log(`
__line);
 }
if(peakActive){
run();   } else {
     if(Ppeak.length > 0 && PpeakCost.length > 0){
                               %s       <%s:%d>:Computing      Peak ,__file,
console.warn(`            parameters`
__func, __line);
     var sumPower = 0;
var sumCost = 0;        var
maxPower = Ppeak[0];       var
maxCost = PpeakCost[0];
var n;      for (let i = 0; i
< Ppeak.length; i++){
var tempPower = Ppeak[i];
if(tempPower > maxPower){
maxPower = tempPower;
     }      var tempCost =
PpeakCost[i];      if (tempCost >
maxCost) {           maxCost =
tempCost;
     }      sumCost += tempCost;
sumPower += tempPower;      n++;
}   var avgPeakCost = sumCost /
n;
                       var avgPeak = sumPower / n;
                       console.warn(` %s <%s:%d>:Max Peak Load
${maxPower}: Maximum Cost ${maxCost}`,__file, __func, __line);
                       console.warn(` %s <%s:%d>:Average peak power
${avgPeak} Average Peak Cost ${avgPeakCost}` ,__file, __func, __line);
                       Ppeak.length = 0;
               }


        }
} var run = async
function(){
       console.warn(` %s    <%s:%d>:in    peak ,__file, __func, __line);
                 period`
       var demand = await computeDemand();
        stream.write(demand +" " +new Date().toISOString() +"\n");
       //stream.end();
       Ppeak.push(demand);
       PpeakCost.push((demand * 15 / 60 * 1000) * Ecost)
       console.warn(` %s                ,__file, __func, __line);
                <%s:%d>:Running`
       // Recycle Loads
     cycleLoads(demand);
     // or shed Loads
       shedLoads(demand);
}
// we can implement a critical shedding scheme that will
// shed high power low priority load first. function
getLoadToShed() {

     var sheddingLoad = activeLoads[0];         for
(let j = 0; j < activeLoads.length; j++){
     var load = activeLoads[j];
           if(sheddingLoad.priority == load.priority) {          // we
will shed the load that consume most power of the two
                       if(load.power > sheddingLoad.power) {
                               sheddingLoad = load;
                       }
               }
               else {
                 if(sheddingLoad.priority > load.priority) {
                 sheddingLoad = load
                       }
               }
```

```
         }
        console.log(` %s <%s:%d>:Load to shed:
${sheddingLoad.name}`,__file, __func, __line);
         return       .ngLoad;
} function setActiveLoads(){        // get
active loads      for (let i = 0; i <
loads.length; i++){
                 var load = loads[i];
            var loadInfo = apputils.find(load, activeLoads);
            if(!loadInfo.state)  {                if(load.state ==
true)  {                      // get active loads
            activeLoads.push(load)
                          }
                 }
         }
        console.log(`%s <%s:%d>:Active load:
${activeLoads.length}`,__file, __func, __line);
}
var shedLoads = async function(demand){
     while(demand > peakCapacity){
                 console.log( %s          <%s:%d>:Threshold ,__file, __func,
                              reached`
__line);

            var lowLoad = getLoadToShed();
     resetLoads(lowLoad);
            var appInfo = apputils.find(lowLoad, recycleLoads);
     if(!appInfo.state)  {
     recycleLoads.push(lowLoad);
                 }
                 demand = await computeDemand();
         }
} var cycleLoads = async function(demand){     // remove load after
recyclyng  while((demand < peakCapacity) && (recycleLoads.length >
0)){
                 console.log(` %s <%s:%d>:load to recycle:
${recycleLoads.length}`,__file, __func, __line);
            setLoadPower(getLoadToRecycle())
     demand = await computeDemand();
         }
} function getLoadToRecycle()
{
     var lowest = 0
     var j = 0;
     var recycleLoad = recycleLoads[0];         for
(let i = 0; i < recycleLoads.length; i++){

                 load = recycleLoads[i];

            if(recycleLoads.priority ==  load.priority ){
     if(recycleLoads.power <= load.power)  {
                                recycleLoads = load;
                          }
                 }
```

```
                else {
                        if(recycleLoads.priority >  load.priority) {
                                recycleLoad = load;
                        }
                }
        }
        return recycleLoad;
}  function remove(load,
loads){
     return loads.filter(item => item !== load) }  function
updateLoadState(load) {        var url='http://localhost:8000/api/oic/'+
load.locationId + '/' + load.name +'?di='+ load.deviceId;
     console.warn(`%s <%s:%d>: About to change appliance ${load.name}
at ${url} state`,__file, __func, __line);
     apputils.postResource(url, load).then(function(result){
process.send({ msg: "dsm load updated", load: load});
        }, (err) => {
               console.error(` %s         <%s:%d>: ,__file, __func,
                               ${err}`

__line);

        });
}  function resetLoads(lowLoad){    console.log(`%s
<%s:%d>:Lowest priority load is:
${lowLoad.name}` ,__file, __func, __line);
        lowLoad.state = false;
     lowLoad.power = 0;
     updateLoadState(lowLoad);     activeLoads =
remove(lowLoad, activeLoads);
}   function
setLoadPower(lowLoad){
        var power = 0;
     for (let i = 0; i < loadsRatings.length; i++){
           load = loadsRatings[i];
     if(load.name ==  lowLoad.name ){
     power = load.rating;                   console.log(`%s
<%s:%d>:Load power:
${load.rating}` ,__file, __func, __line);
                        break;
                }
        }
     lowLoad.state = true;    lowLoad.power
= power;    updateLoadState(lowLoad);
        recycleLoads = remove(lowLoad, recycleLoads);
}
setInterval(() => {
main();
}, timeStep);
```

Appendix E. Observing Service Source Code

```
var request = require('request');
var apputils = require("./app_utils");
require('magic-globals');

var proto = null;
var path = require('path');
var fs = require('fs');
var ca = null;
var args = process.argv.slice(2);
var options = {
    help: false,
host: "localhost",
port: 8000,      https:
false,      obs: false
};
proto = require('http');

const okStatusCode = 200; // All right var
reqOptions = {   host: options.host,   port:
options.port,   agent: new
proto.Agent({keepAlive: true}),   headers: {
    Connection: "keep-alive",      'Content-
Type': 'application/json'
  },   ca: ca } var loads =
[]; var startObserving =
false;

process.on('message', (msg) => {

  if(msg != null) {    console.log(`%s <%s:%d>:Message
from Main process:
${msg.length}`,__file, __func, __line);
        for(let i = 0; i < msg.length; i++){
                loads.push(msg[i]);
        }
       console.info(`%s <%s:%d>: Init Obrseving process with:
${loads.length} to observe`,__file, __func, __line);
if(!startObserving) {
         console.info(`%s                    <%s:%d>:`,__file, __func,
                       Observing.........`
__line)
      observeAppliances();
startObserving = true;
    }
  }
}); function
observeAppliances() {
   for(let i = 0; i < loads.length;
i++){
 var load = {   location:     loads[i].location,
locationId:   loads[i].locationId,    applianceId:
loads[i].applianceId,   name:      loads[i].name,
deviceId:         loads[i].deviceId,       state:
loads[i].state,
               current:     loads[i].current,
               power:       loads[i].power
        };
    retrieveResources(load,onResource, (options.obs = true));
  }
}  function onResource(load, data) {      var loadInfo =
apputils.find(load, loads);  if(loadInfo.state) {
     if(Math.abs((loads[loadInfo.pos].current - data.current) /
1000.0) > 1.0) {
             loads[loadInfo.pos].power = data.power;
loads[loadInfo.pos].state = data.state;
loads[loadInfo.pos].current = data.current ;
process.send({ msg: "Load updated", load: loads[loadInfo.pos]});
               }
        }
}  function retrieveResources(load, callback, observe) {
reqOptions.path = "/api/oic/" + load.locationId + "/" + load.name +
"?di=" + load.deviceId;
if (observe) {
    reqOptions.path += "&obs=1";
  }
  var json = "";
  resourceCallback = function(res) {
          'data', function(data) {
res.on(
      if (observe) {
        callback(load, JSON.parse(data));
                     %s        <%s:%d>: ,__file, __func, __line);
console.info(`        ${data}`
      }
      else {
json += data;
      }
    });
          'end', function() {
res.on(
      if (json)
        callback(load, JSON.parse(data));
    });

       'abort', function() {
res.on(
                                                                  );
console.log(   "event: abort"
    });

  }   var req = proto.request(reqOptions,
resourceCallback);

       'error', function(e) {
req.on(
                                                                           ,
console.log(  "HTTP Request error: %s"                             e.message);
  });


  req.end();
```

```
}
```