Article)

Iotivity Cloud-enabled Platform for Energy Management Applications

Yann Stephen Mandza [1]

[1] Department of Electrical, Electronics and Computer Engineering, Cape Peninsula University of Technology, Bellville, Cape Town, South Africa; 211007242@mycput.ac.za, mandza.y.s@gmail.com
* Correspondence: 211007242@mycput.ac.za

Abstract: In developing countries today, population growth and the penetration of higher standard of living appliances in homes 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 dawn of the smart grid concept, embedded systems, and ICTs have paved the way to novel HEMS design. In this regard, the Internet of Things (IoT), an enabler for smart and efficient energy management systems is seeing increasing attention for optimizing HEMS design and mitigating its deployment cost constraints. In this work, we propose an IoT platform for residential energy management applications focusing on interoperability, low cost, technology availability, and scalability. We focus on the backend complexities of IoT Home Area Networks (HAN) using the OCF IoTivity-Lite middleware. To augment the quality, the servicing, and reduce the cost and complexities of deployment, this work leverages open-source Cloud technologies from Back4App as Backend-as-a-Service (BaaS). This architecture provides consumers and Utilities with a data communication platform within an experimental study illustrating time and space agnostic "mind-changing" energy feedback, Demand Response Management (DRM), and appliance operation control via a HEM App via an Android smartphone.

Keywords: Internet of Things; IoTivity; HEMS; HAN; Cloud; Backend-as-a-Service (BaaS); RTOS, Contiki-OS

1. Introduction

The growing energy consumption in South Africa causes a serious problem to energy supply sustainability. This situation is particularly alarming in domestic places from which originate a great share of peak loads and energy wastage. The recent rolling blackouts and electricity price increase only highlighted this reality and call for sustainable measures to reduce overall consumption [1]. 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 [2].

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 as well as an automation platform for implementing energy management strategies [3]. However, traditional HEMS, designs suffer interoperability, granularity, reusability, scalability, and computing power constraints as well as cost-related limitations that have impeded their performance and restricted their penetration in domestic space [4].

Nevertheless, the advent of IoT, cloud computing, and related technologies mitigate these factors and open novel opportunities in HEM design and deployment. Furthermore, the reduction in the size of embedded systems, as well as the proliferation of networking equipment in living spaces, offers the opportunity to build sophisticated and cost-effective Home area Networks (HAN).

Focusing on the "everything connects to each other" principle, IoT bridges the semantic gap regarding the ubiquity of IoT networks devices [5]. It mitigates the cost and device resources limitations as well as the cost factor that restrict HEM real-world application. However, IoT is not a stand-alone paradigm; rather it is an assortment of several technologies working alongside it. In recent times the interaction between IoT devices and the surrounding environment has expanded thus generating a huge amount of data to be handled [6].

The resource-limited nature of IoT demands expensive hardware and software to store the bulk amount of data [7]. To avoid these limitations, data communications in IoT leverage Cloud-based computing infrastructures to accommodate Big Data requirements, provide protocols translation, data abstraction, and higher computing power (Figure 1) [8]. Lately, numerous research ventures have focused on combining Cloud Computing and IoT to provide users improved services accessible anywhere at the same time.

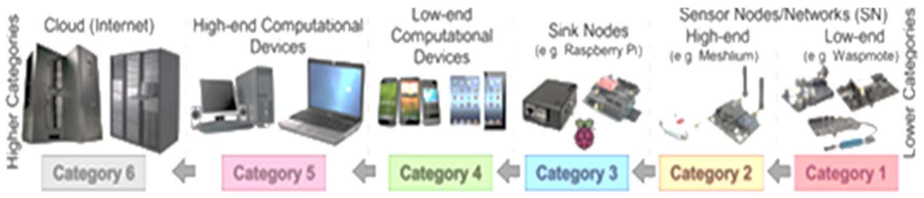


Figure 1. IoT Heterogeneity and computing constraints

As stated in [10], the Cloud "promises high reliability, scalability, and autonomy" for future IoT applications. That is, cloud-based platforms make anything accessible in a time and space agnostic manner favoring user-friendless and performance through backend services(computing, storage, connectivity) or BaaS [11].

Currently, studies on IoT middleware have shown that issues related to devices interoperability, network scalability, and devices management within HEM networks can be greatly alleviated thus offering real-world deployment to HEMS. As stated in [7], middleware in the IoT shall address internet and things issues, handling semantics gap, context awareness, device discovery, manage devices resources, big data, and privacy.

2. Related Works

The IoT is a novel ICT paradigm showing interest from various studies regarding IoT platforms for HEM. The authors in [12] 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 is aimed at decreasing the system deployment cost and facilitating future upgrades, and reconfiguration.

Nevertheless, this setup is archaic and incurs a scalability issue added to the fact that the connection 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.

In [16], the authors 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. Toward this objective, the authors proposed, a UHG responsible for data collection and transmission to the Cloud via the network layer. Openfire, middleware on the gateway server provides the pub-sub mechanism to push information to subscribers. This alleviates the need for private DNS at the consumer's side greatly mitigating implementation constrained and cost.

Nonetheless, the XMPP used within their platform is TCP-oriented (heavyweight) and is expensive for lower-end devices. Moreover, XMPP is part of the IETF standard for IoT. Openfire essentially lacks functionalities such as discovery, provisioning. In [8], is proposed the software architecture for efficient and secure energy management within the smart grid. At the heart of their platform is the IoT gateway (raspberry-PI) running on the Eclipse Kura framework, a free and scalable framework built on Java/OSGi used as middleware to offer hardware interoperability.

Moreover, Cloud connectivity is done through the Mosquito MQTT enabling 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 this work, less attention is given to devices and resource provisioning, discovery, and security. Furthermore, the middleware being used required higher category hardware (see Figure 1) that can run the Java Virtual machine (JVM). This has the drawback of mitigating the expected miniaturization of IoT implementation, increasing cost, and limiting scalability in developing contexts.

In [6], 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 mitigating the cost associated with computing and communications devices, software stack. Thus, they focused on using popular, open-source hardware within their HAN, in this regard, The Raspberry PI acts as a home gateway and TelsoB mote running TinyOS communicating over Zigbee.

To support device-to-device communication, security, and device management, the Author used the Devices Profile for Web Services (DPWS) middleware to abstract the management of HAN devices and provide web connectivity. However, the Web interface provided relies on local router DNS info. These DNS constraints restrict operations to the local network and increase the cost of implementation for remote operations.

[13] proposed an energy management Cloud platform based on the SaaS Cloud model to enhance interoperability via the use of a universal intelligent energy management gateway based on a free Internet of Things (IoT) framework named IoTivity. Here, the authors 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 HAN. Therefore, the authors provided a completed architecture that handles the platform requirement for data communication and device management from appliances on the HAN to services in the Cloud. However, because IoTivity is CoAP based framework, the authors proposed a REST framework for bridging CoAP to HTPP to access their dedicated Cloud infrastructure.

In [14], the authors proposed a framework for energy management applications running on a home gateway offering energy services for multi-homes running on Azure Cloud using dedicated 3rd party providers. Each home is incorporate a gateway (Intel(R) Core (TM) i3-2100 CPU) powered with Microsoft Lab of Things (HomeOS) middleware. The authors provided an Android mobile terminal and Web dashboard using publish/subscribe MQTT model and azure push notification for front-end interfacing. Nevertheless, being a gateway dedicated middleware, LoT limits the miniaturization of IoT HAN devices, thus increasing implementation costs.

2.1. *Research Challenges and Concept Overview*

In summary, the implementation of the HEM platform poses the following major issue:

- Semantic gap (interoperability) and interactivity amongst various manufacturer and communication protocols.
- Miniaturization and performance of IoT devices for seamless penetration of HEM platform.
- Middleware for hardware abstraction, device management, device discovery, connectivity, scalability, adaptability, services customization, and security.
- Cloud platform either as BaaS to export processing and add computing power, remote connectivity, and services.
- Cost of implementing the HEM platform, hardware availability, and protocol stack.

2.2. Research Contributions

To address the above-mentioned challenges, our HEMS platform employs:

- Open-source Middleware using free and accepted IoT protocol stack scalable to lower-end hardware providing disaggregated interoperability, scalability, adaptability, HAN device management, discovery provisioning, and seamless device-to-device(D2D) connectivity and security.
- Leverage, ubiquitous and low–cost and low-power embedded devices as HAN node enhancing technologies availability and penetration in developing context.
- Open-source cloud computing and storage as Software-as-a-Service and BaaS offsetting heavy computation and storage to cloud infrastructure as well providing service for two-way connectivity via subscriptions mechanism and free APIs to both local gateways and mobile device for end users providing integration and control that is both time and space agnostic.

3. Cloud-enabled IoTivity Platform

Novel platforms for HEM should incorporate the features and requirements defined (1) to be economical, affordable to ease their penetration in living places. In this regard, a cloud-based energy management platform leveraging the OCF IoTivity-Lite middleware is proposed. This platform's purpose is to provide energy management based on the Software-as-a-Service (SaaS) model for deploying energy services. The architectural structure leverages the BaaS model for backend computing, data storage, and communication services. The proposed system relies on standard protocols, open-source services, and software and hardware.

An overview of the system architecture is depicted in Figure 2. It uses a three-layer platform consisting of a HAN that gathers consumption data and control appliances, a home gateway to manage devices, and cloud connectivity. A Cloud layer for heavy computing, data storage, and remote services over the third-party tools. 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 Utilities.

.

3.1. Hardware Architecture

The hardware of the HEM platform comprises multiple device categories.

- Though the traditional HEM platform disaggregates devices in terms of connecting, sensing, actuating, and communicating, today's advances in the embedded system allow having all these functionalities in one package within the HAN known as a network node. Our platform uses the popular Arduino AVR&ARM and ESP32 hardware as central slaves processing units.

- Gateway: Communication within IoTivity middleware is mainly IP-based (WIFI and Ethernet). The gateway thus offers protocol translation from the Local COAP based HAN to HTTP/S-based cloud connectivity either as BaaS/SaaS providing the platform with pub-sub(Live-Query) mechanism.
- Computing: The devise that store, process, and analyze data within the platform. Low-level computation is performed at the HAN servers or nodes. However high-level computation and storage are distributed between the local gateway and the Back4App BaaS infrastructure for performance and processing disaggregation.

Figure 2 shows the hardware architecture used for two-way communication, data acquisition, processing, and storage within the proposed platform. As described in 1, our novel platform incorporates all state-of-art, open-source IoT Enabling technologies for higher-end HEM deployment.

Although each node can be interacted with outside the Local COAP Network, Miniaturization and cost-effectiveness required the gateway to primary handle all non-local resource and data requests and computation acting as a proxy server for a local node to remote clients.

3.2. Software Architecture

A certain number of technologies facilitating IoT application development and deployment in smart homes were adopted. Adopting state-of-the-art solutions, we focus on open-source software technology to alleviate the complexity of proprietary software and the related cost.

Resource server nodes sensing, and actuating firmware leverage open real-time operating system (RTOS). This research used the Contiki OS, a bundle whose memory footprint is scaled to fully run-on different Arduino architecture (AVR & ARM), thus adding more consistency, scalability, modularity, and compactness to the local server firmware.

3.2.1. HAN Middleware

The platform handles local networks' interoperability, scalability as well device management complexities using the OCF-IoTivity framework. IoTivity is a communications framework for IoT enabling smooth peer-to-peer connectivity amongst devices irrespective of the underlying OS or protocol satisfying several requisites of the internet of things [8].

Therefore, IoTivity eases the home area network management by handling 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 for devices within category 3 (Figure 1). Thus, porting was developed to support lower category devices.

3.2.2. IoTivity-Lite Arduino Port

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 November 2021 doi:10.20944/preprints202111.0162.v1

To sustain the goal of low cost within the platform, we decided to port the IoTivity-Lite framework to lower category devices (category 1&2). In this regard, we targeted the popular Arduino MCU and the Espressif ESP32 Wi-Fi MCU. 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, we considered two OS, mainly FreeRTOS and ContikiOS. First, we consider these two as being popular RTOS for low power, low-cost MCU. However, we used the ContikiOS on the Arduino MCU because of its low memory footprint and simplicity in developing firmware that is seamless to IoTivity-Lite integration using ContikiOS itself within its stack. In the case of the ESP32 (~500Kbytes of RAM), we adapted an Initial IoTivity port based on the FreeRTOS OS.

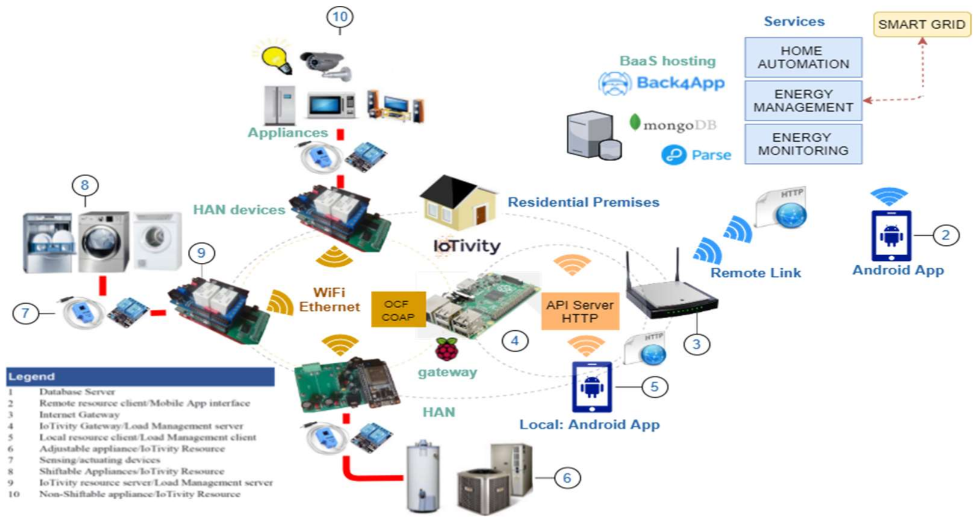

10
Appliances
Services
SMART GRID
BaaS hosting
Back4App
HOME AUTOMATION
ENERGY MANAGEMENT
ENERGY MONITORING
mongoDB
Parse
8
HAN devices
Residential Premises
IoTivity
HTTP
Remote Link
Android App
2
9
WiFi Ethernet
OCF COAP
API Server HTTP
7
4
3
gateway
5
HAN
Local: Android App
6
Legend
1 Database Server
2 Remote resource client/Mobile App interface
3 Internet Gateway
4 IoTivity Gateway/Load Management server
5 Local resource client/Load Management client
6 Adjustable appliance/IoTivity Resource
7 Sensing/actuating devices
8 Shiftable Appliances/IoTivity Resource
9 IoTivity resource server/Load Management server
10 Non-Shiftable appliance/IoTivity Resource

Figure 2. Cloud-based IoTivity-Lite platform HEM architecture

3.2.3. Gateway to Local HAN server interaction

The OCF IoTivity group avails a JavaScript port of the IoTivity stack running on the Node engine or IoTivity-node for the high-level device. Using the IoT-rest-API-server, a NodeJS REST server for HTTP-based communication leveraging IoTivity-node as a client for the local CoAP network devices.

Thus, we used the IoTivity-node empowered IoT-rest-API-server on the gateway device (Raspberry PI) to manage discovery and resources access within the IoTivity-Lite network.

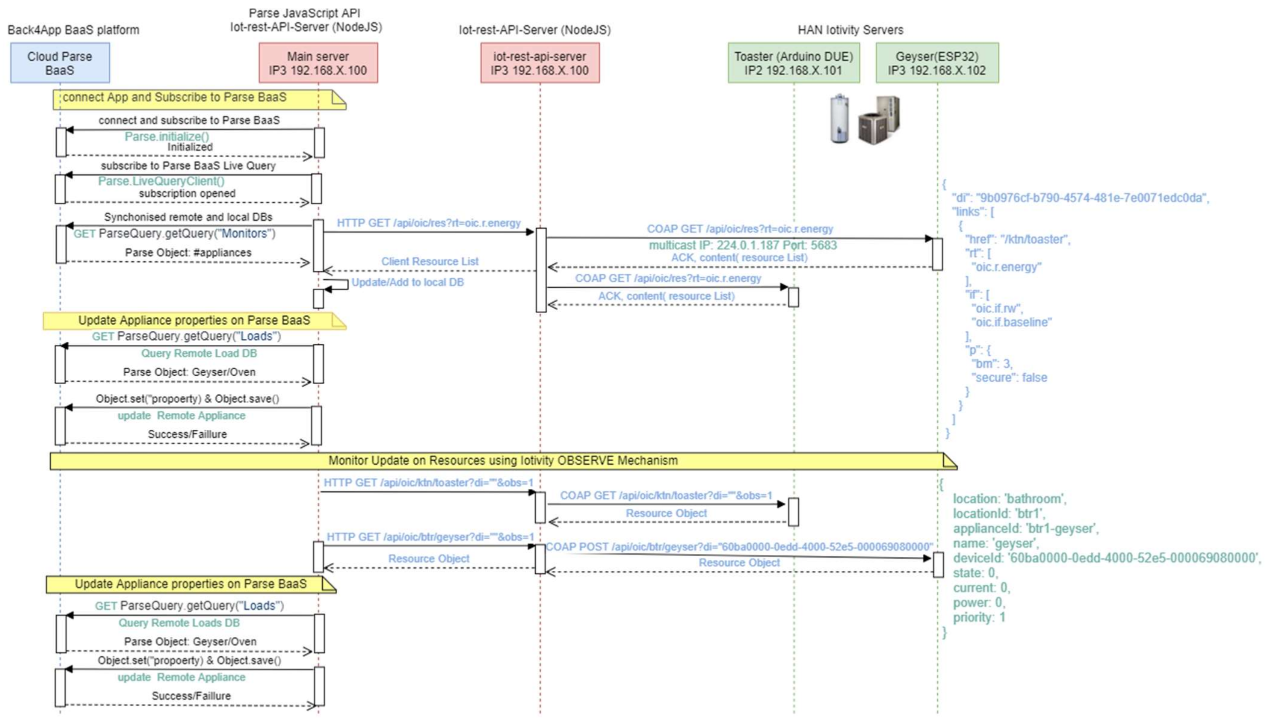
3.2.4. Gateway to Local HAN server interaction

In this work, the Cloud is mainly used as Backend-as-a-service (BaaS), offering the pub-sub mechanism (Live Query) offering subscriptions service on the platform data. It provides storage, backend service related to computation, feedback Home Automation (HA) services. We used Parse an open-source server providing a RESTful API for a plethora of devices and OS on the different programming languages (We use the JavaScript API).

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 no native Parse server environments and lack important Parse BaaS tools. In this regard, we used the back4App Cloud platform 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 on a mobile or web application.

3.3. Communication Architecture

Local network configuration and data communication within the platform is presented in Figure 3. The IoTivity-Lite platform integrates with existing network (WIFI and Ethernet) infrastructure in the residential place. All IoTivity-Lite servers have dynamically assigned IP addresses on the private network 192.168.0.X starting at 192.168.0.100 for the Gateway (Raspberry PI).


Back4App BaaS platform
Cloud Parse BaaS
Parse JavaScript API
Iot-rest-API-Server (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
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
COAP GET /api/oic/res?rt=oic.r.energy
Update/Add to local DB
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 3. Local and Cloud configuration and communication in the Platform

When powered on, the IoT-rest-API-server provides the gateway with HTTP/HTTPS access to IoTivity-Lite HAN slave servers. Next, is the “main server” process initialized connectivity to the Cloud BaaS infrastructures on Back4App Cloud? On the cloud BaaS, we create an App, databases table, namely, “Loads” storing appliances data, “SmartHomes” storing all registered smart home,” DRM” storing DRM data for each smart home (Figure 4). To scale this deployment to multiple smart homes, we implemented an “owning” relationship amongst the different classes.

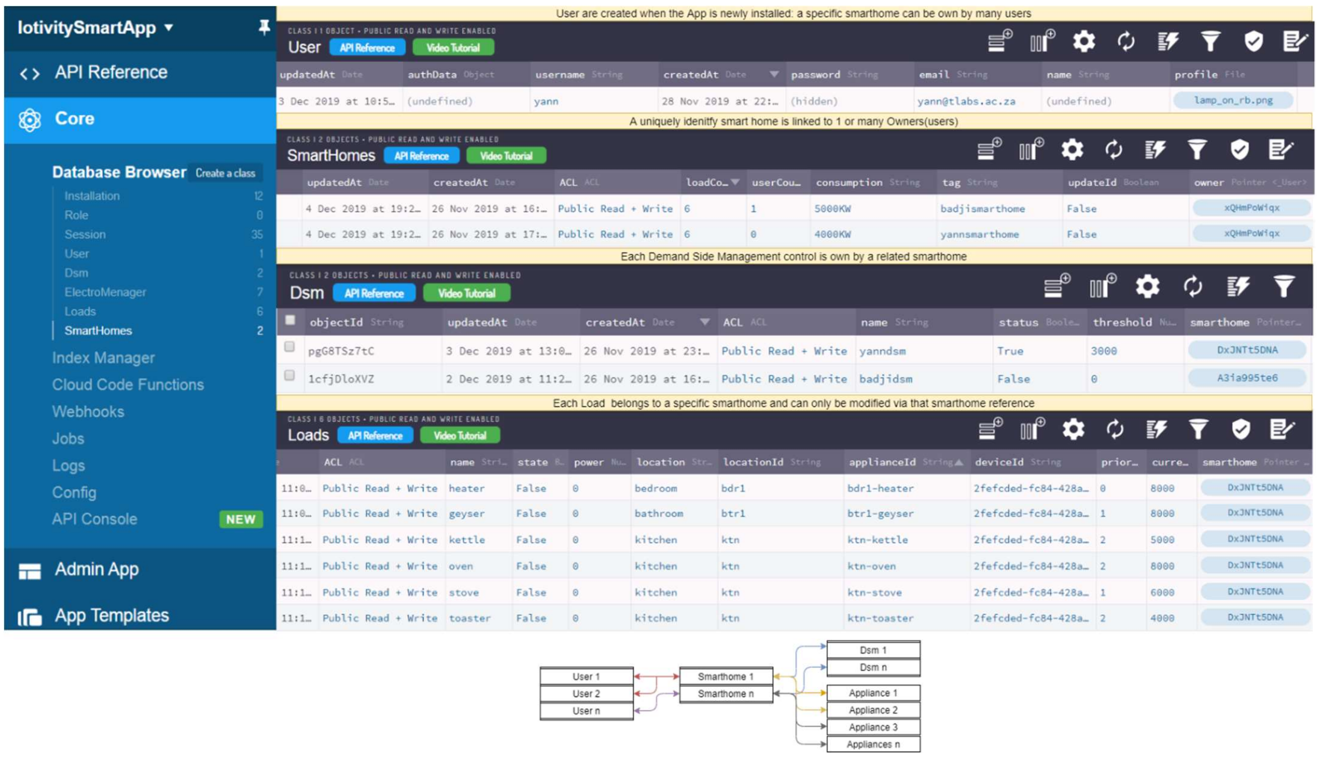
This mechanism scales the system by enabling many owners in the “User” table to own a specific “smart home” that owns many different appliances in the “Loads” table and a specific “DRM” object (Figure 4). Moreover, this method allows us not to create a class/table for each smart home context, thus keeping all related data together, easing development and maintenance. Following a user login/signup process, the “main server” starts a pub-sub subscription to the related “smart home”, DRM and Loads resources on the Cloud platform using the Parse Server Live Query mechanism (Figure 3).

This connection is realized by authenticating 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. It enables each client endpoint to receive events on the entry in the subscription list.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 November 2021 doi:10.20944/preprints202111.0162.v1

Amongst other events emitted within the subscription are the "create" and "update" events which are received in real-time by the subscribed client alongside meta-data regarding the specific entry that was created/updated. We initialized the Parse server Live Query mechanism using our Back4App App application ID, JavaScript Key, and its Master Key for authentication purposes.

An important feature of the Parse server on the Back4App platform is the cloud code functionality. This tool enables the developer to run NodeJS functions directly on the Back4App Cloud. This step immediately makes the Cloud Code Functions available to the IoT platform and can be used to implement services (Utility DRM portal). After configuration, the gateway server initiates a login/signup sequence with the Cloud user authentication services.



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	A3ia995te6

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. Platform databases structure and connections on Back4App BaaS

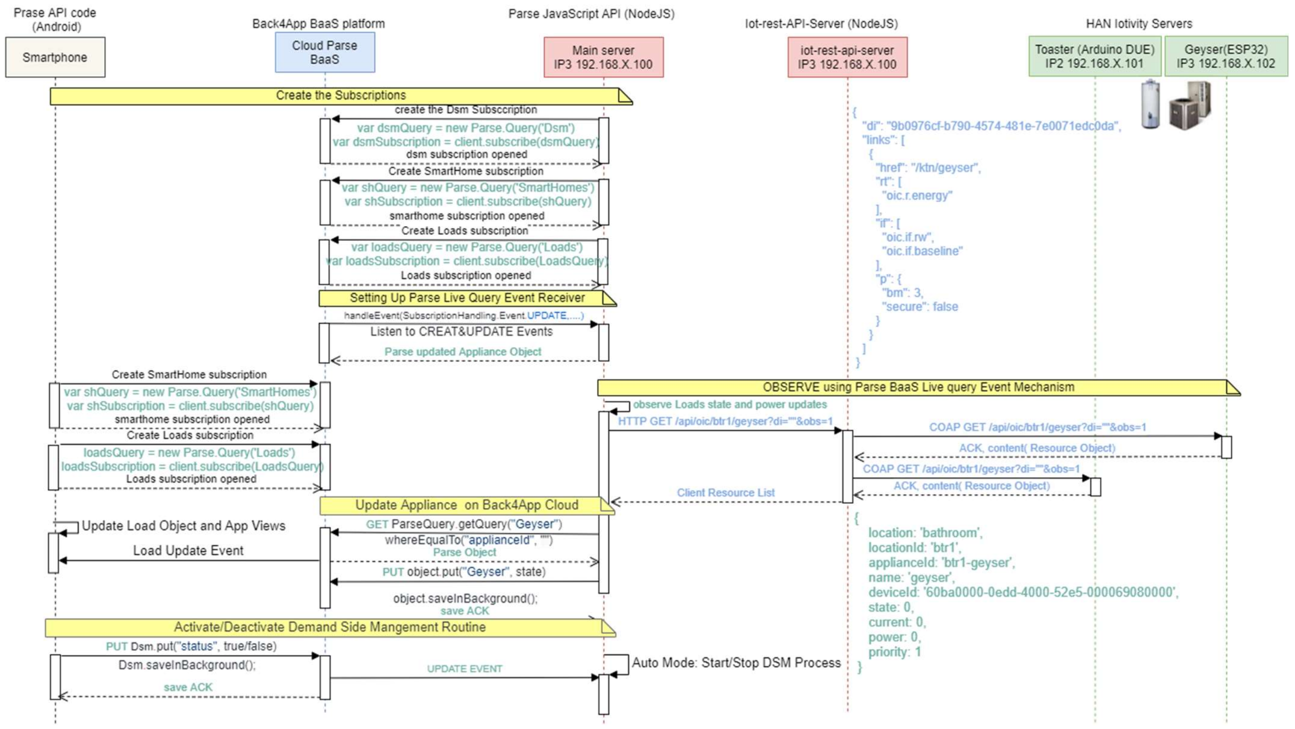
Next, the gateway server initiates the local DBs (monitor and Load tables are created if not existing already) for offline storage (we used MySQL DB engine). The offline storage is synchronized with an initial DB query to the online storage which returns the provisioned number of loads. This process is two-fold.

First, the gateway server sends a DB query to get the number of known and provisioned appliances and retrieve those from the local storage which is also able to store newly discovered appliances. Secondly, resource discovery 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. After initialization, an observation service on the IoTivity network using the OBSERVE mechanism is started and resources properties are regularly updated.

When a mobile client uses the energy, App participates in the platform information exchange, first, the App establishes a connection to the cloud backend and starts a client subscription (Figure 5). After the client is successfully login/signup as an authenticated user, we make use of the Back4App BaaS security features on the client and on the home gateway side to provide secured data communication in both ways. 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. The gateway server thus generates an HTTP POST request to the IoT-rest-API-server which generates a CoAP POST (i.e., /api/oic/ktn/kettle?di='') with the new state (i.e., state: true/false) that 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 process and updates the App front-end.


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
Update Load Object and App Views
Load Update Event
GET ParseQuery.getQuery("Geyser")
whereEqualTo("applianceId", "")
Parse Object
PUT object.put("Geyser", state)
object.saveInBackground();
save ACK
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 5. communication flow with remote/local Android smartphone App

4. Experimental Results on Case Study

To evaluate the performance of our cloud-based IoTivity platform in addressing the challenges of HEM design, we implemented an experimental setup. Leveraging an energy application on an Android smartphone tested the platform's performance in providing real-time energy monitoring and home automation (HA). We then implemented a peak load management DRM algorithm to manage consumption at home. The HEM platform was deployed for the resistive load (appliances) as shown in Figure 6. The detailed specification for the hardware used is listed in Table 1.

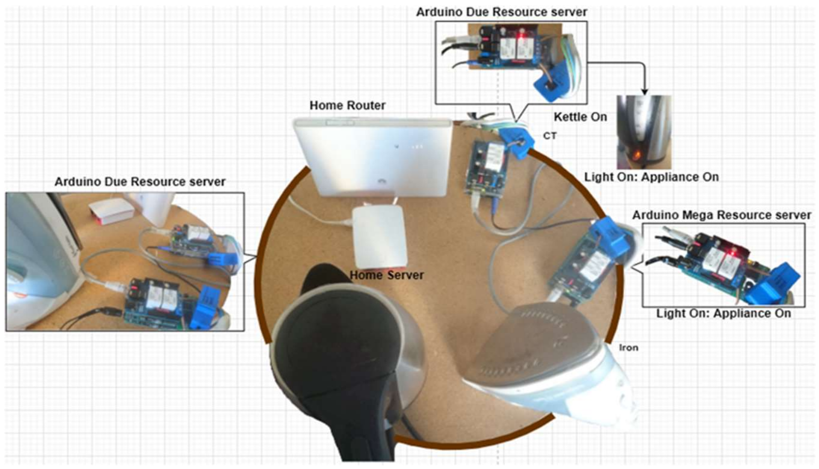

Arduino Due Resource server
Home Router
Kettle On
CT
Light On: Appliance On
Arduino Due Resource server
Arduino Mega Resource server
Home Server
Light On: Appliance On
Iron

Figure 6. Experimental setup under evaluation

The HAN's devices are augmented with motes designed and manufactured as plug-and-play shields. These shields provided the sensing and actuating interface to existing home appliances via non-invasive and safe electronics devices a DAQ shield on each mote is embedded with sensing and actuating electronics for 1Vrms CT sensor signal output and 30A AC actuating relays.

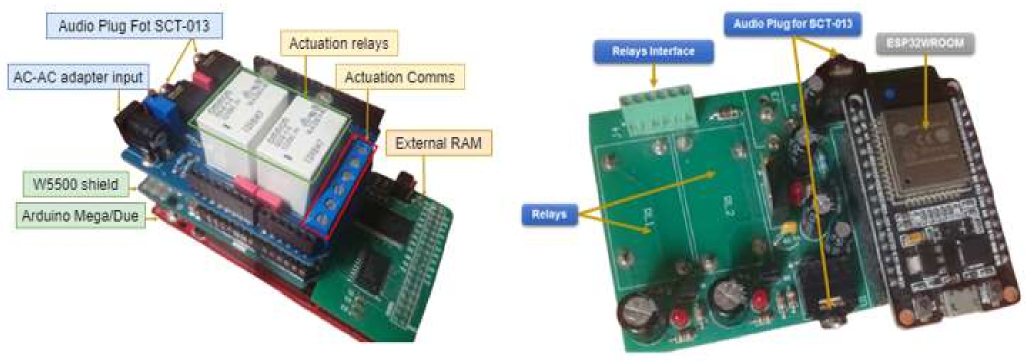
Communication within the HAN for the Arduino-based mote is Hardwire(Ethernet using the Wiznet 5500 ethernet shield), whereas WIFI is used the ESP32 based motes.

Table 1: Experimental setup hardware used for the home area network

Devices	Model	Processors	Operating System
Arduino	Mega 2560	Atmega 2560	ContikiOS
	ARM(DUE)	32 bits SAM3X8E ARM Cortex-M3	
ESP32	ESP-WROOM-32	Xtensa Dual-Core 32-bit LX6 MCP	FreeRTOS
Ethernet Shield	2nd Generation	Wiznet W5500	N/A
Raspberry PI	3rd Generation Model B+	64-bits BCM28374 ARM Cortex-A53, 1.2 GHz	32 Bits Raspbian Stretch

The AVR-based mote is augmented with an external memory bank to improve its performance in handling the secured deployment of the iotivity-Lite stack. 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. This code is used by the higher-level server code within the GET and PUT methods.

The power properties for an appliance connected to either a mote based on the Arduino or ESP32 MCU were computed based on the algorithm that samples the current and voltage for 25 cycles (at 50Hz) or 500ms to calculate the different root mean square properties and accumulate those to calculate the power consumption. We use a 10bits ADC setting on Arduino AVR and 12bits on ARM and ESP32.



(a) (b)

Figure 7. Iotivity HAN servers (appliances interface nodes): (**a**) AVR/ARM DAQ node prototype; (**b**) ESP32 node DAQ Prototypes.

4.1. Experimental Results

In this section, we present the results of the case study focusing on the observation of all scenarios executed to establish our platform performance. Using the setup in Figure 4, we test the feedback and home automation scenario within the platform. That is, we present the response from the IoTivity-Lite HAN server device. That is the Arduino and ESP32 slaves' response to resources requests.

Then, we show the underlining software services handling the smart-home local and remote connectivity. In this regard, describe the different initialization steps via curtailed logs of each of the services running on our raspberry PI local home server. Secondly, we present feedback results, that is home consumption real-time, and enhance visualization anytime anywhere via the energy App.

Thirdly, we demonstrate home automation using the Energy App to turn home appliances on/off. Lastly, we detailed the DRM scenario condition and assumption and show the result of our peak shaving algorithm.

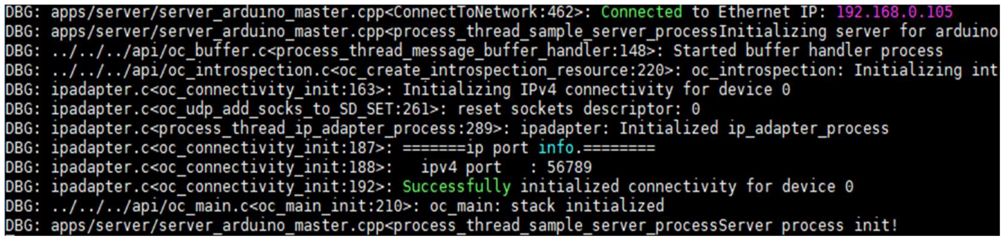

```
DBG: apps/server/server_arduino_master.cpp<ConnectToNetwork:462>: Connected to Ethernet IP: 192.168.0.105
DBG: apps/server/server_arduino_master.cpp<process_thread_sample_server_processInitializing server for arduino
DBG: ../../../api/oc_buffer.c<process_thread_message_buffer_handler:148>: Started 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 8. HAN slavers Initialisation logs

The firmware on the resource server on the HAN runs the IoTivity-Lite core that was ported to the AVR and ARM Arduino Arch. In figure 8 above, we see 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.

4.1.1. Feedback via Energy App

Regarding the Energy monitoring scenario, we evaluated the platform's ability to provide space agnostic, real-time feedback. Using a Sony Xperia Z5 smartphone we tested Energy feedback on our platform using the IotSmartApp as shown in Figure 6.

The energy consumption is presented in engaging visual tools both graphic and textual with compelling colors (red under the consumption curve). The evaluation shows that feedback can be dispatched via the platform within ~3 seconds from HAN to the Back4App BaaS then to the smartphone App.

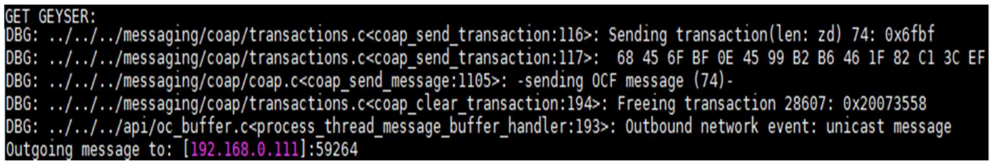

```
GET GEYSER:
DBG: ../../../messaging/coap/transactions.c<coap_send_transaction:116>: Sending transaction(len: zd) 74: 0x6fbf
DBG: ../../../messaging/coap/transactions.c<coap_send_transaction:117>:  68 45 6F BF 0E 45 99 B2 B6 46 1F 82 C1 3C EF
DBG: ../../../messaging/coap/coap.c<coap_send_message:1105>: -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 9. HAN server GET response

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. Figure 9 shows the logs of GET requests received from the slaves, followed by the IoTivity-Lite stack processing of the request and a response (74 bytes of resource data) to the client on 192.168.0.111:59264.

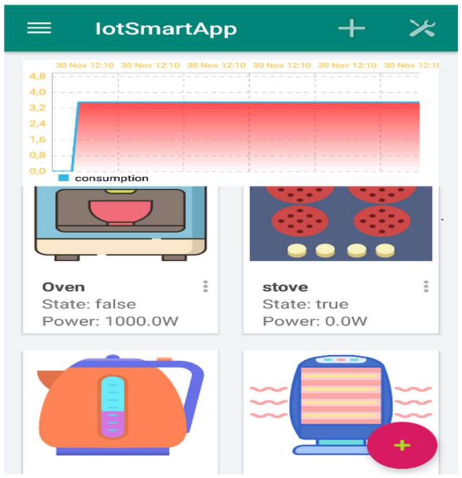
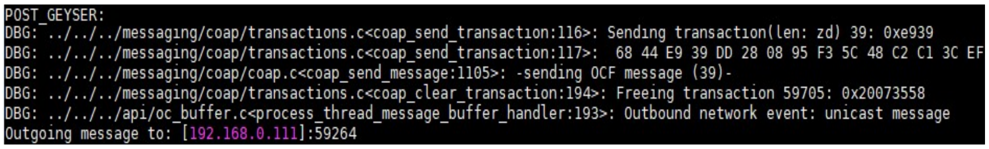


Figure 10. Energy Consumption Monitoring on IotSmartApp

4.1.2. Home automation via Energy App

We evaluated the ability of our platform to provide space agnostic on/off control of the home appliances under consideration. In Figure 11, a POST interaction is executed whenever the client request and update an appliance status (On/Off).

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 11. HAN server POST response

In Figure 12, we present feedback about home automation via our 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). On (b) the iron is turned on (lamp is yellow), the consumption (power) at that instant was recorded as 1.16 KW. This is practically demonstrated in Figure 6, the Arduino server connected to the physical appliance control circuit is activated (red light is on).

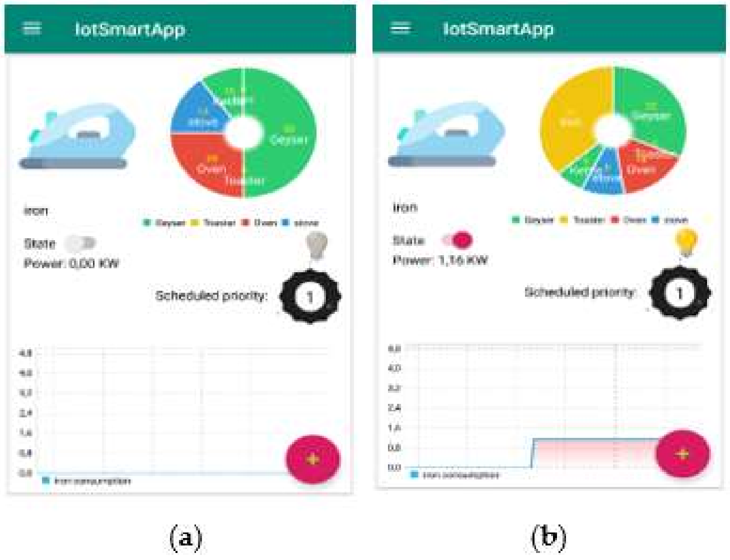


Figure 12. Home Automation via the IoTSmartApp; (**a**) Appliance is turned off; (**b**) Appliance is turned on.

4.1.3. DRM via Energy App

We implemented a DRM algorithm for peak load management as a service that aims to demonstrate the impact of our platform on residential load efficiency. We followed related works in the area of HEM to define our experimental model.

To demonstrate the performance of their IoT architecture for residential load, the author in [15], implemented an experiment based on a maximum allowable peak threshold of 33KW. 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.

The author in [6], implemented a smart transformer control-as-a-service over Fog computing limiting the load of each home at 4KW. The algorithm monitors the status of the power source and activates a DR signal when overload by cycling all homes and shedding load in a home that has exceeded the 4KW thresholds. In both studies, the demonstrated DRM does not consider user preferences. In [16], the authors introduced three-level

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 November 2021 doi:10.20944/preprints202111.0162.v1

priority scheduling for home appliances so users can switch on home appliances subject to their satisfaction level and preferences. Peak load DRM depends on mathematical models. In this case study, we focus on regularly operated or fixed appliances and develop an algorithm based on equations from works [17].

However, a default value of 5KW based on literature and the appliances of Table 2 will be used by the algorithm that will be implemented. Figure 13 depicts the experimental platform and the data flow for this case study.

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 600Kwh/months. Municipality regulated rated the unit in this context at 278.46 c/kWh (City of Cape Town, 2019).

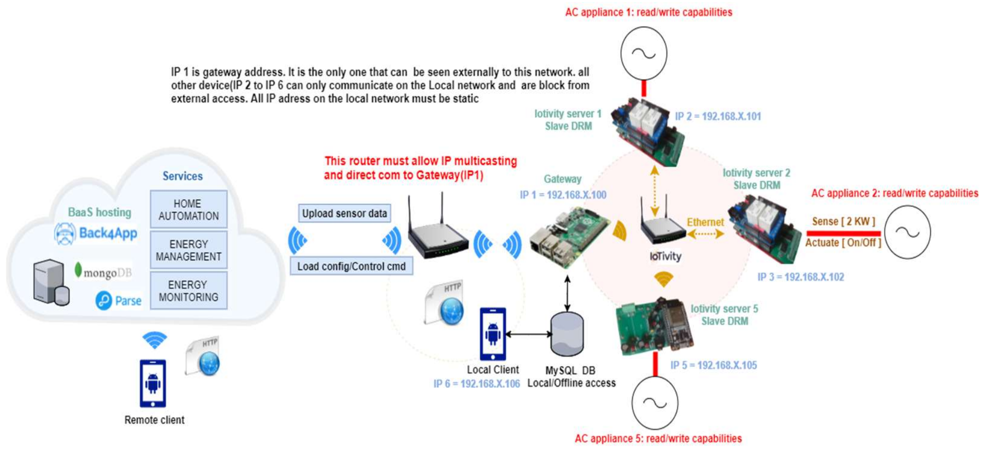
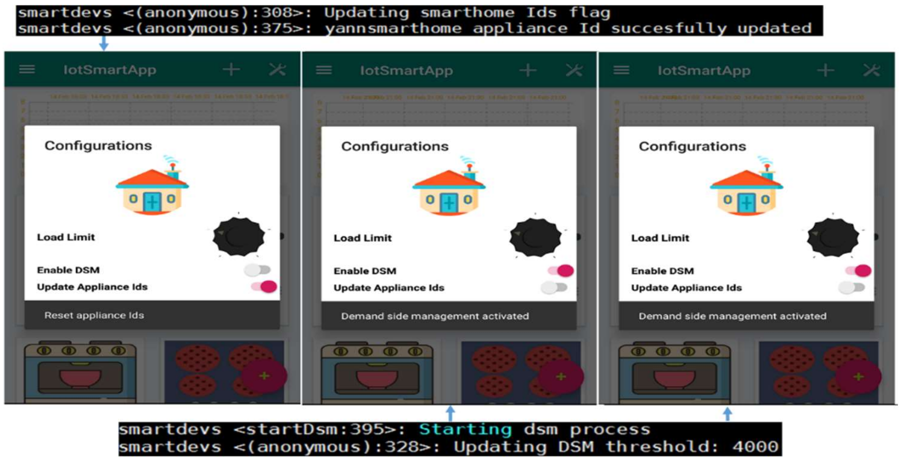

AC appliance 1: read/write capabilities
IP 1 is gateway address. It is the only one that can be seen externally to this network. all other device(IP 2 to IP 6 can only communicate on the Local network and are block from external access. All IP adress on the local network must be static
Iotivity server 1 Slave DRM
IP 2 = 192.168.X.101
This router must allow IP multicasting and direct com to Gateway(IP1)
Services
BaaS hosting
Back4App
HOME AUTOMATION
ENERGY MANAGEMENT
ENERGY MONITORING
mongoDB
Parse
Upload sensor data
Load config/Control cmd
Gateway
IP 1 = 192.168.X.100
Iotivity server 2 Slave DRM
AC appliance 2: read/write capabilities
Ethernet
IoTivity
Sense [2 KW]
Actuate [On/Off]
IP 3 = 192.168.X.102
HTTP
Iotivity server 5 Slave DRM
Local Client
IP 6 = 192.168.X.106
MySQL DB
Local/Offline access
IP 5 = 192.168.X.105
Remote client
AC appliance 5: read/write capabilities

Figure 13. Case study system architecture

The DRM scenario was tested via our developed Energy App. In Figure. 14, configuration windows are proposed to the user to set up the current DRM algorithm threshold, reset the smart home's appliance IDs, and activate/de-activate the DRM service running on the gateway server. When the user activates the DRM service, both the new status and threshold are passed to the listening home server via the Parse 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 is 5KW with a 10% positive margin (5.5KW) based on appliances properties and priority setting in Table 2. At the end of each peak period, we calculate 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 guiding equation (3) for digitized for the DRM algorithm is presented in Appendix A.



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

Reset appliance Ids

Demand side management activated

Demand side management activated

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

Figure 14. DRM with Energy App

We sampled appliances consumption at 10 min duration. However, a timeframe of 5 min was used for peak simulation running the algorithm at 10s interval then normalizing the results. 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. The red line shows the maximum allowable demand threshold (about 5.5KW). When the demand exceeds the peak limit, the DRM service turns some appliances off according to the priority assigned. We can see the DRM load profile (brown curve) peak is lowered and the valleys are filled as expected of a peak shaving algorithm.

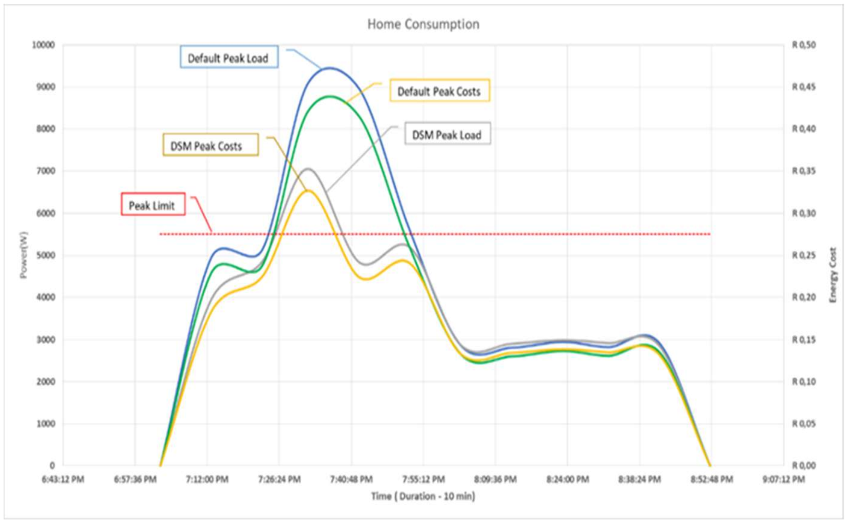


Figure 15. Peak load profiling via IoTivity HEM platform

5. Discussions

The main highlights from the experimental setup concerning the research objectives:

- IoTivity-Lite Middleware: The IoTivity Middleware from OCF was selected to handle interoperability, scalability, and resource management semantic gaps inherent in IoT systems. Experimentation showed that indeed both WIFI, Ethernet devices could effectively and uniformly exchange data through the IoTivity-Lite HAN. Though the essential functions of the IoTivity-lite middleware are effective; functionalities such as provisioning and security are only available n Arduino DUE and ESP32 due to AVR board limited RAM. Latencies in data delivery of ~4 seconds were observed for Gateway-to-cloud communicating over the Back4App cloud services;

- Porting IoTivity-Lite Arduino AVR&ARM: The IoTivity middleware is a recent on-going project with a growing community and interest, however, this framework was not available on low-memory, low-cost hardware such as the Arduino architecture, therefore, porting the IoTivity Middleware to Arduino was performed in this work and represented one of the novelties of this thesis. For this, Contiki OS was used and adapted for Arduino Arch, the experimentation shows that Contiki on OS on Arduino is stable and responsive and its memory footprint is lightweight enough to allow enough space on the Arduino RAM for IoTivity stack features such as discovery, CRDUN operations, device, and resource provisioning on AVR arch as well security on DUE devices:

From the above observations, further work and focus should be placed on.

- 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
- 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, and thus easily implement all security mechanisms available. This also reduces the development load and facilitates maintenance.
- Wireless communication, here WIFI should be adapted to all HAN devices for easier penetration and adaptation in residential places. We recommend to used technology with embedded wireless protocol to optimize the HAN data communication.
- Smart grid signals from the Energy utility can take advantage of this platform. but the interface needs to be fully defined from the cloud Backend this can be a cloud Job provided as SaaS responding to request from the Utility

6. Conclusions

This article strives to participate in the growing research targeting the modernization of the electric grid within the smart grid effort to handle the growing peak demand and limitation of the traditional grid, mainly in the residential sector.

Therefore, “Cloud-based IoTivity platform for Home Energy management applications”, a cost-effective, efficient, and performing communication platform leveraging IoT enabling tools were presented and deployed to provide smart energy management applications particularly in domestic places within the South African context.

The article addressed the IoT semantic gaps regarding interoperability, scalability, and the cost and availability of technology issues related to HEM deployment. We thus focused on the architectural design and backend requirements of the platform around open source IoT technologies and developed a completed prototype providing an experimental setup to test the platform performance for smart-grid-related interventions in households.

.

Author Contributions: All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Supporting data and current regarding the porting of the Iotivity-Lite middleware to Arduino AVR&ARM can be found here: https://github.com/yannS2016/iotivity-constrained-arduino. Home Area Network server running the IoTivity Lite firmware, Parse backend development scripts and Gateway server development files for the experimental platform can be found here: https://github.com/yannS2016/iotsmartapp

Acknowledgments: I acknowledge the support of Dr. Atanda Raji, from the Cape Peninsula University of Technology, my supervisor for the entirety of this work for his support in providing both technical and administrative support.

Conflicts of Interest: The authors declare no conflict of interest

Appendix A

We consider a home focus on fixed consumption load comprises primarily those with resistive load. Based on works from [18], we formulated equations (10 to (3). For this 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. for this model we considered 6 appliances (table 5.4-1). 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 sliced into a horizon time slot series T ϵ {1, 2, 3, . . ., T}. since each peak period span the same time length of $\daleth_{peak}$ (4 hours), considering that each time slot is 15 min long, thus T is a series of 16 elements. The total power consumption during a peak period is expressed as $\varsigma_{A_nT_L}$

$$\varsigma_{A_nT_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \zeta(t) \right) \tag{1}$$

Where $\boldsymbol{P_n(t)}$ is the power consumption for appliance 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

$$£_{A_nT_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \varepsilon(t) \times \zeta(t) \right) \tag{2}$$

Where ε(t) represents the cost of electricity at time t ϵ T.

Based on equation (1), we develop the algorithm for our DRM case study as below

$$\varsigma_{A_nT_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \zeta(t) \right) \le \gamma(t) \tag{3}$$

Where γ(t)is the home threshold? That is γ(t) is the maximum allowable peak load at time t ϵ T. we start with a dynamic value for γ(t) of 5 KW.

References

1. Abu-Mahfouz, A. M.; Olwal, T. O.; Kurien, A.M.; Munda, J.L and Djouani, K. Toward developing a distributed autonomous energy management system (DAEMS). *AFRCON* **2015**, pp.1-6.
2. Numsa, D.; Mudumbe, J. M.; Ndwe, T. The internet of things for a smart South African grid architecture. in Proceedings of the 8th International Development Informatics Association Conference **2015**, pp. 95–107.
3. Blanco-Novoa Ó, Fernández-Caramés TM, Fraga-Lamas P, Castedo L. An Electricity Price-Aware Open-Source Smart Socket for the Internet of Energy. *Sensors*. **2017**; 17(3):643, https://doi.org/10.3390/s17030643.
4. Vine. D.; Buys, L.; Morris, P. The Effectiveness of Energy Feedback for Conservation and Peak Demand: A Literature Review. OJEE **2017**; pp. 7–15, https://doi: 10.4236/ojee.2013.21002.
5. Wang, F.; Hu.; L.; Zhou, K.; Zhao, K. A Data Processing Middleware Based on SOA for the Internet of Things. Journal of Sensors. **2017**, 2015, pp. 1–8, https://doi.org/10.1155/2015/827045.
6. Al Faruque, M. A.; Vatanparvar, K. Energy Management-as-a-Service Over Fog Computing Platform, IEEE IoT-J 2016, 3, pp. 161-169, https://doi: 10.1109/JIOT.2015.
7. Lin, H.; Bergmann, N.W. IoT Privacy and Security Challenges for Smart Home Environments. Information 2016, 7, 44. https://doi.org/10.3390/info7030044.
8. Beligianni,F.; Alamaniotis, M.; Fevgas, A.; Tsompanopoulou , P.; Bozanis, P.; Tsoukalas, L. H. An internet of things architecture for preserving privacy of energy consumption MedPower 2016, pp. 1-7, http://doi.10.1049/cp.2016.1096.

9. Vinh, T. Le.; Bouzefrane, S.; Farinone, J. M.; Attar, A.; Kennedy, B. P. Middleware to integrate mobile devices, sensors and Cloud computing, Procedia Computer Science. Elsevier Masson SAS 2015, 52, pp. 234–243
10. Risteska Stojkoska, B. L.; Trivodaliev, K. V. A review of Internet of Things for smart home: Challenges and solutions, *Journal of Cleaner Production* **2017,** 140, pp. 1454–1464.
11. Khatu, M.; Kaimal, N.; Jadhav, P.; Rizvi, S. Implementation of Internet of Things for Home Automation, IJEERT, 3(2), pp. 7–11.
12. Sagar, V.; Kusuma, S. M. Home Automation Using Internet of Things, IRJET, 02, 2015, pp. 56–72.
13. Lee, C.; Lai, Y. H. Design and implementation of a universal smart energy management gateway based on the Internet of Things platform, *ICCE* **2016,** pp. 67-68, http://doi.10.1109/ICCE.2016.7430524.
14. LI, X.; Nie, L.; Chen, s.; Zhan, D.; Xu, X. An IoT Service Framework for Smart Home: Case Study on HEM, IEEE International Conference on Mobile Services, 2015, pp. 438-445, http://doi.10.1109/MobServ.2015.66.
15. Viswanath, S. K.; Yuen, C.; Tushar, W.; Li, W. T.; Wen C. K. System design of the internet of things for residential smart grid, *IEEE Wireless Communications* **2016,** pp. 90-98**,** htpp://doi.10.1109/MWC.2016.7721747
16. Rasheed, M. B, Javaid, N.; Ahmad, A.; Awais, M, Khan, Z. K.; Qasim, U.; Alrajeh, N. Priority and delay constrained demand side management in real-time price environment with renewable energy source, *International Journal of Energy Research*, **2016**, 40, pp. 2002-2021, https://doi.org/10.1002/er.3588.
17. Hussain HM, Javaid N, Iqbal S, Hasan QU, Aurangzeb K, Alhussein M. An efficient demand side management system with a new optimized home energy management controller in smart grid, Energy 2018; 11(1):190. https://doi.org/10.3390/en11010190
18. Serov, A. N.; Makarychev, P. K. Evaluation of the Effect of Nonlinearity of the Successive Approximation ADC to the Measurement Error of RMS, (INDEL), 2018, pp. 1-6, doi: 10.1109/INDEL.2018.8637630.