IoT

MDPI

Article

IoTivity Cloud-Enabled Platform for Energy Management Applications

Yann Stephen Mandza * and Atanda Raji

Department of Electrical, Electronics and Computer Engineering, Cape Peninsula University of Technology, Bellville, Cape Town 7500, South Africa; rajia@cput.ac.za
* 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 overall consumption and peak load. The dawn of the smart grid concept, embedded systems, and ICTs have paved the way for novel Home Energy Management Systems (HEMS) design. In this regard, the Internet of Things (IoT), an enabler for intelligent and efficient energy management systems, is the subject of 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 addressed the backend complexities of IoT Home Area Networks (HAN) using the Open Consortium Foundation (OCF) IoTivity-Lite middleware. To augment the quality, servicing, reduce the cost, and the development complexities, this work leverages open-source cloud technologies from Back4App as Backend-as-a-Service (BaaS) to provide 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) under a peak shaving algorithm yielded peak load reduction around 15% of the based load, and appliance operation control using a HEM App via an Android smartphone.

Keywords: internet of things; IoTivity; HEMS; HAN; cloud; Backend-as-a-Service; RTOS; contiki

Citation: Mandza, Y.S.; Raji, A. IoTivity Cloud-Enabled Platform for Energy Management Applications. *IoT* **2022**, *3*, 73–90. https://doi.org/10.3390/iot3010004

Academic Editor: Hyun-Ho Choi

Received: 8 November 2021
Accepted: 17 December 2021
Published: 27 December 2021

1. Introduction

The growing energy consumption in South Africa constitutes a significant impediment to energy supply sustainability. This situation is particularly alarming in domestic areas, which make up a growing share of peak loads and energy wastage. The recent rolling blackouts and electricity price increase only highlighted this reality and calls 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 energy utility initiatives to an 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 necessary for the grid to extend its technological tools to residential buildings. HEMS provides consumers with feedback on appliances or equipment operation and offers an automation platform for implementing energy management strategies [3]. However, traditional HEMS suffer several design and architectural limitations. Indeed, constraints related to device interoperability, data granularity, system reusability and scalability, computing power, and hardware cost have impeded HEMS performance and restricted their penetration in households [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 size reduction of embedded systems, and 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 network devices [5]. It mitigates the cost and device resources limitations and cost factors that restrict HEM real-world applications. However, IoT is not a stand-alone paradigm. Rather, it is an assortment of several technologies working together. In recent times the interaction between IoT devices and the surrounding environment has expanded and generated an enormous 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.

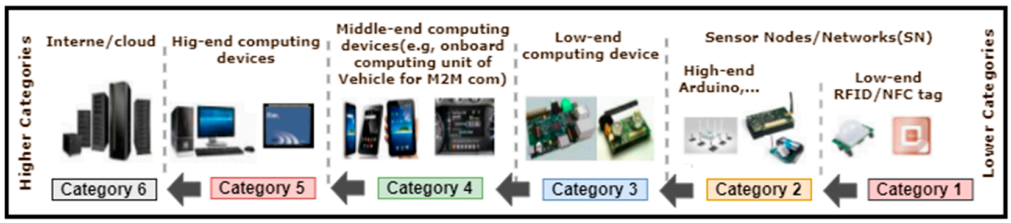


Figure 1. IoT heterogeneity and computing constraints.

As stated in [9], the cloud "promises high reliability, scalability, and autonomy" for future IoT applications. That is, cloud-based platforms support connectivity to things. Such platforms make anything accessible in a time and space agnostic manner, thus favoring user-friendliness and performance through backend services (computing, storage, connectivity) or BaaS [10]. Currently, research on IoT middleware has shown that IoT middleware can greatly alleviate issues related to device interoperability, network scalability, and device management within HAN networks. Thus, IoT middleware favor real-world HEM applications. As stated in [7], middleware in the IoT shall address internet and things issues, handle semantics gap, context awareness, device discovery, manage device resources, big data, and privacy.

2. State of the Art

IoT is a novel ICT paradigm showing promise in various studies regarding IoT platforms for HEM. The authors in [11] 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 goal is to decrease the system deployment cost and facilitate future system 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 the restrictive private DNS in many developing 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 the consumption data.

In [12], the authors propose an IoT platform targeting residential consumers leveraging smartphone and cloud technologies to offer Smart grid empowered energy management (DRM signal) and home automation as services. To this end, the authors proposed a UHG transmitting collected data to the cloud via the network layer using Openfire as middleware on the Gateway to provide the pub-sub mechanism to push information to subscribers. This

architecture alleviates the need for a private DNS at the consumers' side and significantly mitigates implementation constraints and cost.

Nonetheless, the XMPP is TCP-oriented (heavyweight) and expensive for lower-end devices. Moreover, it is unrecognized by the IETF standard for IoT. The Openfire server essentially lacks functionalities such as discovery and provisioning, and security is not supported on lower-end devices, thus increasing the cost. In [8], software architecture is proposed 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 within the platform is enabled using Mosquito MQTT, pushing the stream of smart meter data to a message broker at the edge for analytics and knowledge extraction, and later forwarding the aggregated data securely to the cloud, preserving privacy. In this work, less attention is given to HAN devices management, resources provisioning, device 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 [13], a fog computing-based platform for energy management focusing on interoperability, scalability, adaptability, and local and remote monitoring while leveraging open-source software/hardware enabled users to implement 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 stacks. 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. This limits operation on the local network and increases the cost of implementation when an ISP subscription is required for remote control operations.

In [14], the authors proposed an energy management cloud platform based on the SaaS cloud model to enhance interoperability via the use of a universal smart energy management gateway based on a free Internet of Things (IoT) framework named IoTivity. The author used the IoTivity middleware to abstract from the monolithic, ad hoc implementation that locks traditional HEMS to a private protocol or mechanism, limiting the choice and spectrum of possible HAN. Therefore, a completed architecture that handles the platform requirement for data communication and device management from appliances on the HAN to services provided in the cloud was provided. However, because IoTivity is a CoAP based framework, the authors proposed a REST framework for bridging CoAP to HTPP to access their dedicated cloud infrastructure.

In [15], 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 (Homes) middleware. The authors provided an Android mobile terminal and web dashboard using a publish/subscribe MQTT model and Azure push notification for front-end requirements. Nevertheless, being a gateway dedicated middleware, IoT limits the miniaturization of IoT HAN devices, thus increasing implementation costs.

2.1. Research Challenges and Concept Overview Section

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

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

2.2. Research Contributions

To address the above-mentioned challenges, our novel platform for HEM employs:

- Open-source middleware using free and accepted IoT protocol stack and scalable to lower-end hardware to provide disaggregated interoperability, scalability, adaptability, HAN device management, discovery provisioning, and seamless device-to-device (D2D) connectivity and security.
- Leverage, ubiquitous, low-cost and low-power embedded devices as HAN node, enhancing technology availability and penetration in a 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 as providing service for two-way connectivity via subscriptions mechanisms and free APIs to both local gateways and mobile devices 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 in Section 1 in order to be economical and affordable for the common consumer, and thus increase their penetration in living spaces. A cloud-enabled energy management platform based on the Backend-as-a-Service (BaaS) model built around the OCF IoTivity framework is proposed. The platform's architecture centers on standard protocols and open-source services and software.

Figure 2 below, depicts an overview of the proposed architecture. We thus present a three-layer architecture consisting of a HAN gathering consumption data and controlling appliances. Next a home gateway to manage HAN devices and to handle cloud connectivity. Finally, a Cloud layer for computing, data storage, and remote services over the third-party tools and a smartphone App as user front-end for enhanced feedback. Additionally, the Cloud provides an interface to smart grid services as these are made available by utilities providers.

3.1. 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 slave processing units.
- Gateway: Communication within IoTivity middleware is mainly IP-based based (Wi-Fi and ethernet). The gateway thus offers protocol translation from the Local COAP based HAN to HTTP/S-based cloud connectivity either as BaaS or SaaS providing the platform with pub-sub (Live-Query) mechanism.
- Computing: The devices 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 a two-way communication, data acquisition, processing, and storage within the proposed platform. As described in Section 1, our novel platform incorporates all state-of-art, open-source IoT Enabling technologies for higher-end HEM deployment.

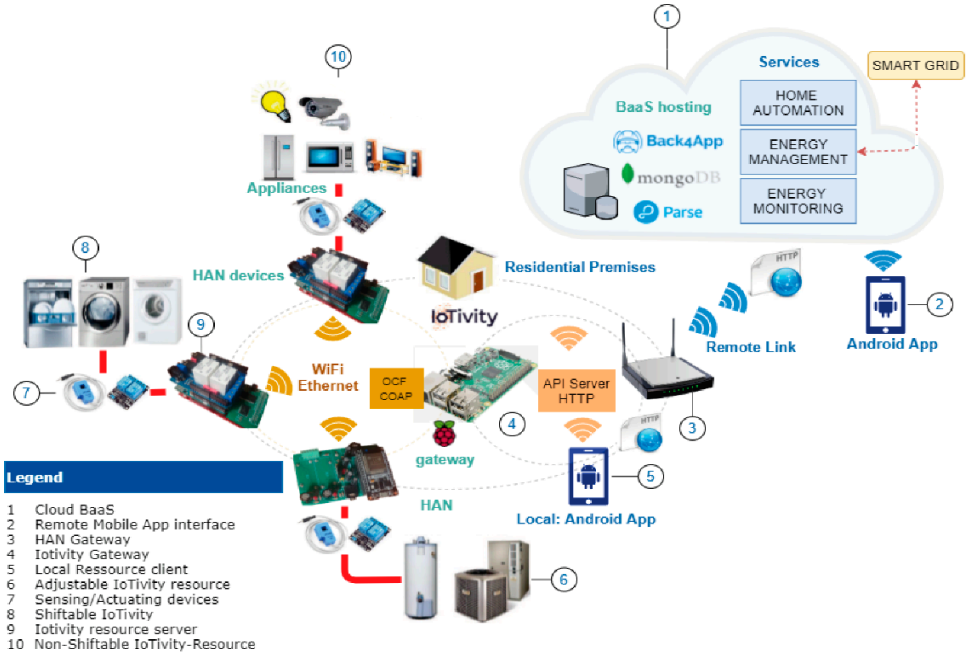


Figure 2. Cloud-based IoTivity-Lite platform HEM architecture.

Each node can be interacted with outside the Local COAP Network. However, miniaturization and cost-effectiveness require the gateway to primarily handle computations and all non-local resources and data requests while acting as a proxy server for the local node to remote clients.

3.2. Software Architecture

Several technologies facilitating IoT application development and deployment for smart homes are adopted to guide experimental development. Adopting state-of-the-art solutions, we focus on open-source software technology to alleviate the complexity of proprietary software and the related cost.

For the sensing and actuating and resource server node, the firmware leverages open real-time operating system (RTOS), namely 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

Our platform will handle local networks' interoperability, scalability, and device management complexities using the OCF-IoTivity framework, 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 hardware within category 3 (Figure 1). Thus, an adaptation or port was developed to support lower category devices on IoTivity-Lite.

3.2.2. IoTivity-Lite Arduino Port

To maintain 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 relies on an OS running on the MCU.

Based on the literature review on RTOS and the state-of-art, we considered two OS, mainly FreeRTOS and Contiki OS. These RTOS are popular for low power, low-cost MCU. However, we used the Contiki OS on the Arduino MCU because of its low memory footprint and simplicity in developing firmware that is seamless to IoTivity-Lite integration using Contiki OS itself within its stack. In the case of the ESP32 (~500 Kbytes of RAM), we adapted an Initial IoTivity port based on the FreeRTOS OS.

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. Cloud Tools and Infrastructure

In this work, the Cloud is mainly used as a Backend-as-a-Service (BaaS), offering the pub-sub mechanism (Live Query) subscriptions service on the platform data. It provides storage, backend service related to computation, and 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 (the JavaScript API was used).

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

In this work, the Cloud is mainly used as a 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, and feedback Home Automation (HA) services. We used Parse, an open-source server providing a RESTful API for a plethora of devices and OS for the different programming languages (We use the JavaScript API).

Parse server is flexible and can be hosted and migrated from one *c*loud 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 important Parse BaaS tools are lacking. In this regard, we used the back 4AppCloud 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.

The IoT-rest-API-server provides the gateway with HTTP/HTTPS access to IoTivity-Lite HAN slave servers when powered on. Next, the "main server" process initializes 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 data, "DRM" storing DRM data for each smart home (Figure 3). We implemented an "owning" relationship amongst the different classes to scale this deployment to multiple smart homes.

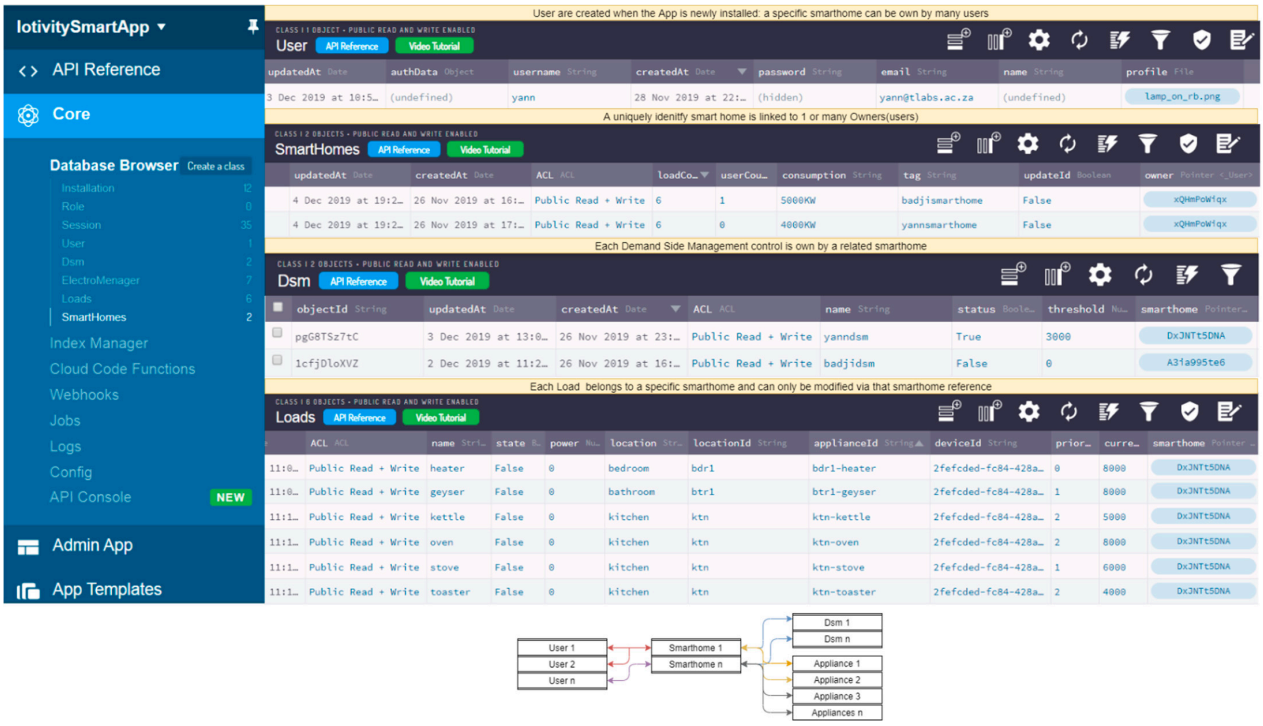


Figure 3. Platform databases structure and connections on Back4App BaaS.

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 4).

This connection is realized by authenticating the user and defining the "smart home" against the username. 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.

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

A useful 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. Cloud code functionality offers energy utilities a backdoor to implement incentives and management programs (Utility DRM portal). After configuration, the gateway server initiates a login/signup sequence with the cloud user authentication services.

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

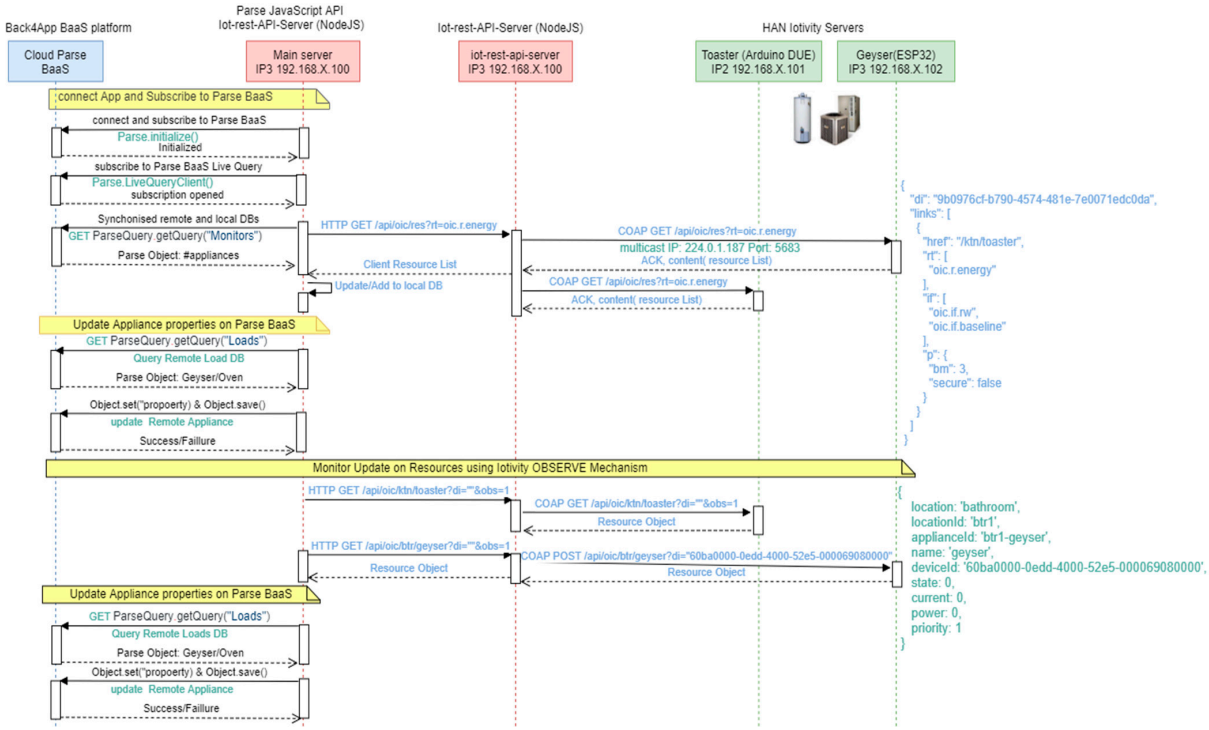


Figure 4. Local and cloud configuration and communication in the platform.

First, the gateway server sends a DB query for the number of known and provisioned appliances. Then, it retrieves those from the local storage that can store newly discovered appliances. Secondly, a resource discovery request is sent to the IoT-rest-API-server service. This server 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 resource properties are regularly updated.

When a mobile client using 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 successfully completes login/signup, Parse server BaaS security features are used on the client and the home gateway to provide two-way secured data communication. All GET requests are submitted as Parse GET queries for each mobile client to access the related homes databases.

Lastly, the remote DB appliance properties are also updated. After initialization, an observation service on the IoTivity network using the OBSERVE mechanism is started and resource properties are regularly updated.

When a mobile client uses energy, the app participates in the platform information exchange. First, the app establishes a connection to the cloud backend and starts a client subscription (Figure 5). The client successfully login/signup as an authenticated user. The system uses the Back4App Parse security features on the client and 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.

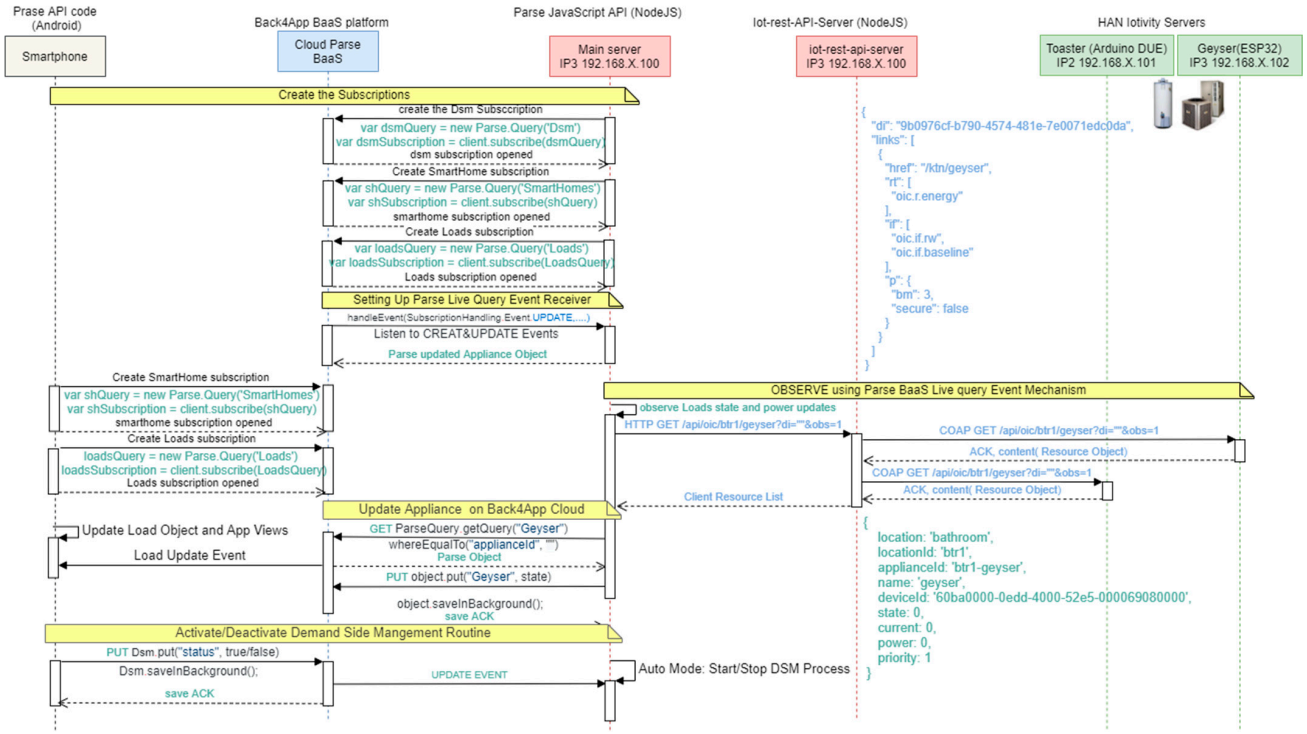


Figure 5. Communication flow with remote/local Android smartphone app.

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.

4. Experimental Results of 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 (appliance) as shown in Figure 6.

The detailed specification for the hardware used is listed in Table 1 below.

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

Devices	Model	Processors	Operating System
Arduino	Mega 2560 ARM (DUE)	Atmega 2560 32 bits SAM3X8E ARM Cortex-M3	Contiki OS
ESP32	ESP-WROOM-32	Xtensa Dual-Core 32-bit LX6 MCP	FreeRTOS
Ethernet Shield	2nd Generation	Wiznet W 5500	N/A
Raspberry PI	3rd Generation Model B +	64-bits BCM28374 ARM Cortex-A53, 1.2 GHz	32 Bits Raspbian Stretch

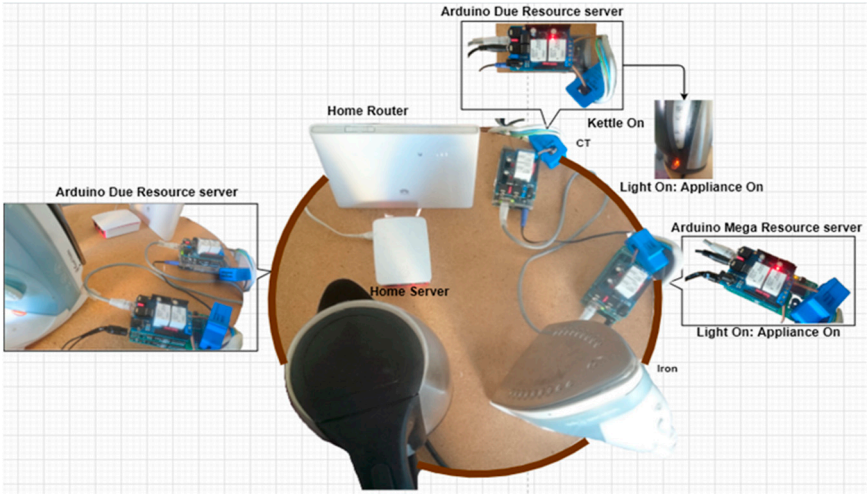


Figure 6. Experimental setup under evaluation.

The HAN's devices or motes are designed and manufactured as plug-and-play shields (Figure 7). These shields provide 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.

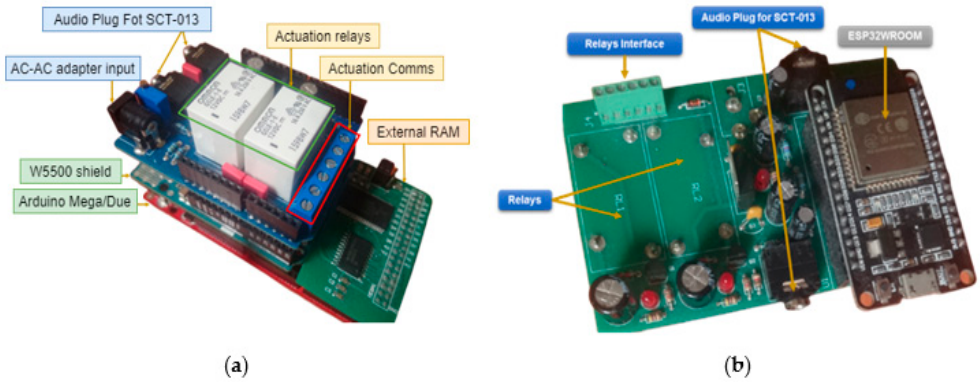


Figure 7. IoTivity HAN servers (appliance interface nodes): (**a**) AVR/ARM DAQ node prototype; (**b**) ESP32 node DAQ Prototypes.

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

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) comprises the IoTivity-Lite server code and the low-level sensing and actuating 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 firmware within the GET and PUT methods.

The firmware calculates from the sensed current and voltage the power properties of each appliance connected to a mote based on the Arduino or ESP32 MCU. This computation is based on the algorithm that samples the current and voltage transformers for 25 cycles (at 50 Hz) or 500 ms to calculate the different Root Mean Square (RMS) power accumulated to compute the power consumption. We used a 10 bits ADC setting on Arduino AVR and 12 bits on ARM and ESP32.

4.1. Experimental Results

In this section, we present the case study results 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 resource requests.

Then, we show the underlying software services handling the smart-home local and remote connectivity. In this regard, describing 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, i.e., home consumption in real-time, and enhanced 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.

The firmware burnt on the HAN resource server runs the IoTivity-Lite core ported to the AVR and ARM Arduino Arch. In Figure 8 below, 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 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 8. HAN slavers initialization logs.

4.1.1. Feedback via Energy App

For the energy monitoring scenario, we evaluated the platform's ability to provide space-agnostic and real-time feedback. 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 resources 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 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.

```
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.

Using a Sony Xperia Z5 smartphone we tested Energy feedback on our platform using the IoTSmartApp as shown in Figure 10 below.

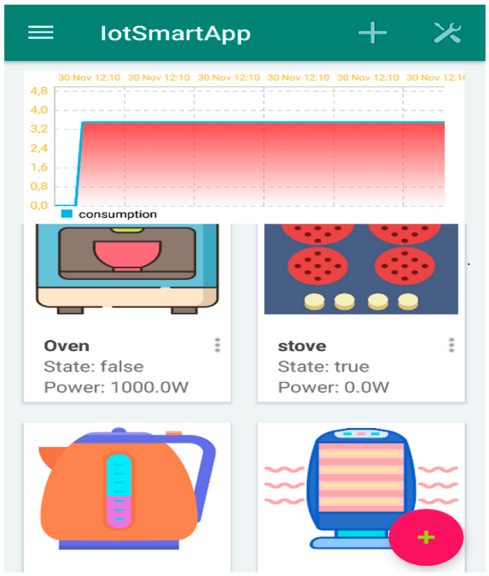


Figure 10. Energy consumption monitoring on IotSmartApp.

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 s from HAN to the Back4App BaaS then to the smartphone App (Figure 10).

4.1.2. Home Automation via Energy App

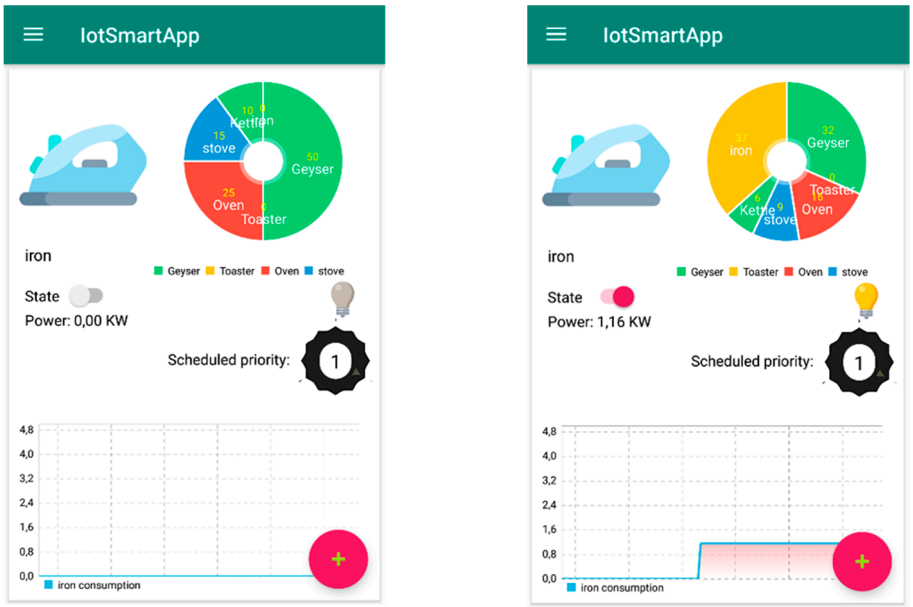
We evaluated the platform's ability to provide space agnostic on/off control of the home appliances under consideration. In Figure 11, a POST interaction is performed whenever the client request and update an appliance status (On/Off).

```
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.

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.

In Figure 12, we present feedback about home automation via our IotSmartApp. This experimentation targets an iron-rated 1200 W within the tested setup. On Figure 12a the iron is off, thus its state is false (the lamp is grey). On Figure 12b the iron is turned on (lamp is yellow), the consumption (power) at that instant was recorded as 1.16 kW. In Figure 6, can practically be seen the Arduino server connected to the physical appliance control circuit in an activated (red light is on).



(**a**) (**b**)

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 around 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 33 kW. The author's algorithm controls light bulbs at each house at peak time, slotting a 24-h time duration into 8640 time periods. That is their smart grid simulation detected the total demand every 10 s. The author in [13], implemented a smart transformer control-as-a-service over fog computing, limiting the load of each home at 4 kW. 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 4 kW thresholds. In both studies, the demonstrated DRM does not consider user preferences. In [16], the authors introduced three-level 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. The case study considers regularly operated or fixed home appliances and develops an algorithm based on appliances operation priority settings and equations from works proposed in [17,18].

The DRM algorithm used a default value of 5 kW based on the literature. Figure 13 depicts the experimental platform and its two-way data transmission.

The algorithm runs with an electricity price per unit considering a household in the research context (City of Cape Town) with consumption equal to or above 600 kWh/month. Municipality regulation rated the power consumption unit at 278.46 c/kWh (City of Cape Town, 2019).

The developed energy app is used to test the DRM scenario. 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 algorithm output for analysis was logged and plotted to appreciate the benefit of the peak-saving algorithm. The maximum allowable peak demand is 5 kW with a 10% positive margin

(5.5 kW) based on appliances properties and priority setting in Table 2 above. We calculate the average power and energy cost, and maximum peak power. This data is made available as statistical info to each smart home user. The guiding Equations (A1)–(A3) digitized for the DRM algorithm are detailed in Appendix A. In Figure 14 below, 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.

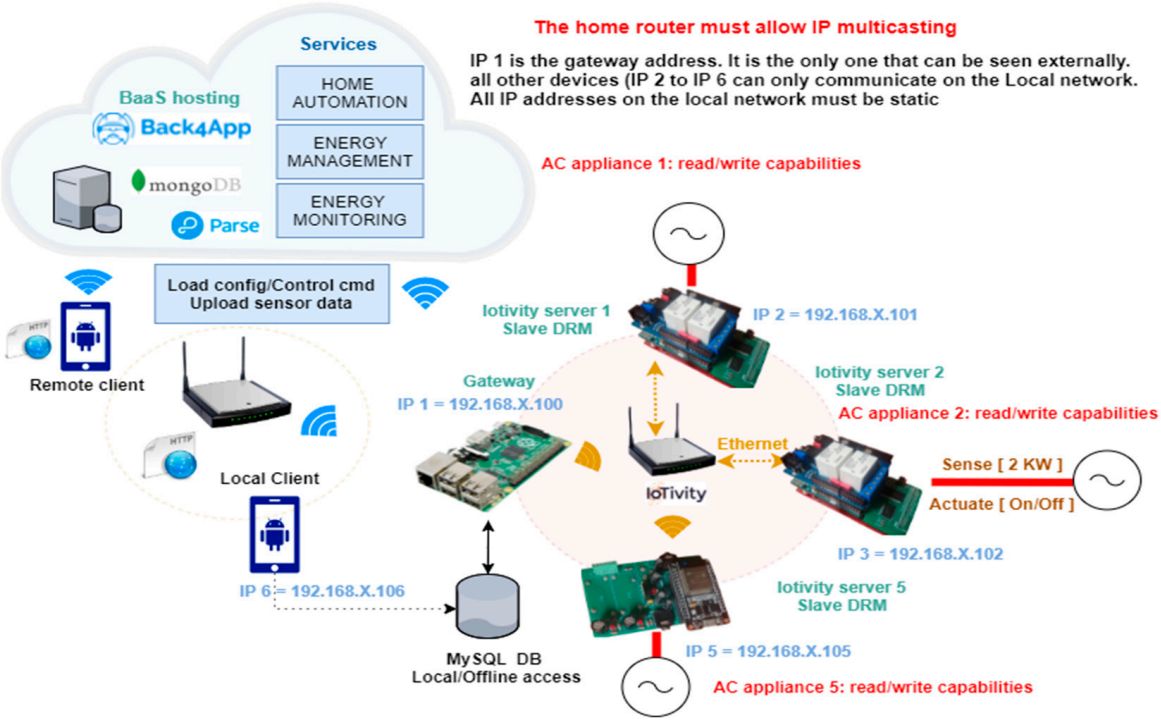


Figure 13. Case study system architecture.

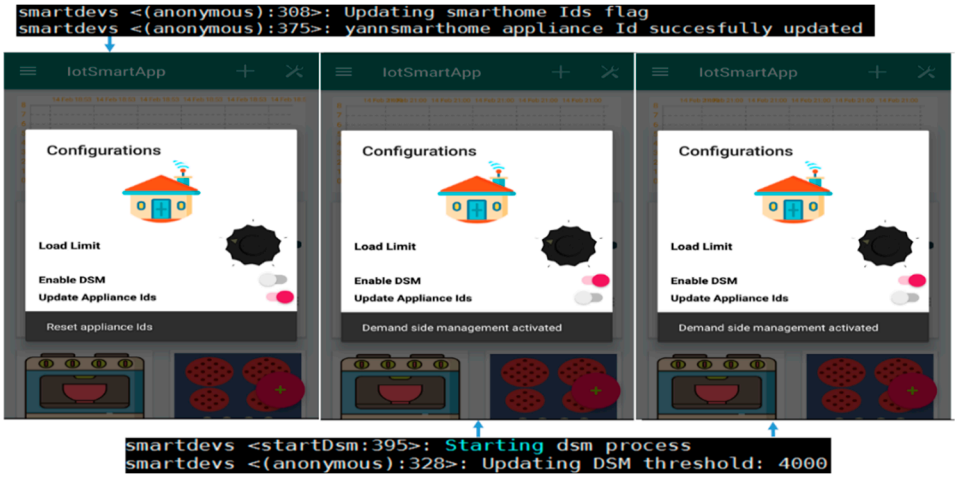


Figure 14. DRM with energy app.

Table 2. Appliances in the considered home with their typical priority level.

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

We sampled appliance consumption at 10 min duration. However, a 5 min timeframe was used for peak simulation, running the algorithm at a 10 s interval then normalizing the resultThe grey and blue curve of Figure 15 denotes 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 5.5 kW). 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.

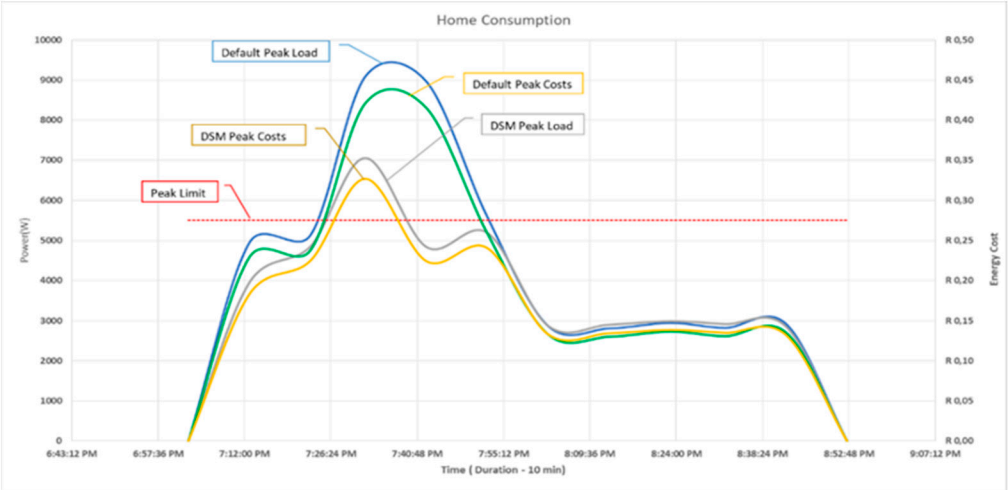


Figure 15. Peak load profiling via IoTivity HEM platform.

5. Discussion

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

1. 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 Wi-Fi and 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 in Arduino DUE and ESP32 due to AVR board limited RAM. Latencies in data delivery of ~4 s were observed for Gateway-to-cloud communicating over the Back4App cloud services.
2. Porting IoTivity-Lite Arduino AVR & ARM: The IoTivity middleware is a recent ongoing 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, the IoTivity middleware is ported to the Arduino MCU representing one

of the novelties of this work. To this end, Contiki OS was used and adapted for Arduino Arch. The experimentation shows that Contiki OS on Arduino is stable and responsive, and its memory footprint is lightweight enough to allow sufficient 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:

3. Enhancing security using the IoTivity onboarding and provisioning mechanism to authenticate the client that interfaces to the HAN resource server. This feature was not fully implemented because of software inconsistency with the IoT-rest-API-server.
4. 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 the IoT-rest-API-server easily implementing all security mechanisms available. This also reduces the development load and facilitates maintenance.
5. Wireless communication, Wi-Fi should be adapted to all HAN devices for easier penetration and adaptation in residential places. We recommend using technology with embedded wireless protocol to optimize the HAN data communication.
6. 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 in response to requests from the utility.

6. Conclusions

This article strives to participate in the growing research concerning the smart grid's potential for modernization of the electricity grid in the effort of energy utilities to effectively handle increasing peak load, especially 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 technologies, was presented and deployed to provide and demonstrate intelligent energy management applications, mainly in domestic places within the South African context. This article addressed the IoT semantic gaps regarding interoperability, scalability, and the cost and availability of technology issues pertaining to HEM.

Thus, we 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's performance for smart-grid-related interventions in households. The experimental DRM load profile shows that the demand promptly falls back below the peak limit after performing the peak shaving algorithm, generating savings of up to 17% on the morning and evening peak loads. The overall response time for GET or POST requests for device-to-device and cloud-to-device resource requests average ~4 s for feedback and appliance actuation.

7. Future Research

The low-cost and miniaturization requirements present noticeable performance issues in terms of hardware memory constraints and response time and limit the deployment of IoTivity middleware security and management tools. Improvement of the platform could focus 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
- A higher-end embedded device for HAN servers able to handle multiple clients while maintaining a fast response time was observed as an issue with AVR motes, and in some capacities with the DUE servers due to its reduced processing speed and constrained memory. A miniaturized, higher memory wireless MCU running at a faster clock would provide a faster response time.

- Smart grid signals from the energy utility can take advantage of this platform. However, the interface needs to be fully defined from the cloud interface. This can be a cloud job that requires monitoring or listening via an API provided by utility-to-smart-grid incentives and propagation of these to home gateways. A more modern approach would be to leverage "Data Lakes" on cloud platforms.

Author Contributions: Y.S.M.: Conceptualization, Literature review, Formal analysis, Methodology, Validating, Writing original draft. A.R.: Conceptualization, review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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. (accessed on 12 December 2021). 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 (accessed on 12 December 2021).

Acknowledgments: I acknowledge the support of 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. For our model, let $A_n \epsilon \{a_1, a_2, a_3, \cdots, a_n\}$, such that $a_1, a_2, a_3, \cdots, a_n$ represents each appliance. For this model we considered 6 appliances (Table 2). 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,$\cdots$, T}. since each peak period span the same time length of T_{peak} (4 h), 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 $St_{A_n T_L}$

$$St_{A_n T_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \zeta(t) \right) \tag{A1}$$

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

$$\pounds_{A_n T_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \varepsilon(t) \times \zeta(t) \right) \tag{A2}$$

where $\varepsilon(t)$ represents the cost of electricity at time $t \in T$.

Based on Equation (A1), we develop the algorithm for our DRM case study as below

$$St_{A_n T_L} = \sum_{t=1}^{T} \left(\sum_{n=1}^{An} P_n(t) \times \zeta(t) \right) \leq \gamma(t) \ (3) \tag{A3}$$

where $\gamma(t)$ is the home threshold. That is, $\gamma(t)$ is the maximum allowable peak load at time $t \epsilon T$. We start with a dynamic value for $\gamma(t)$ of 5 kW.

References

1. Abu-Mahfouz, A.M.; Olwal, T.O.; Kurien, A.; Munda, J.; Djouani, K. Toward developing a distributed autonomous energy management system (DAEMS). In Proceedings of the AFRICON 2015, Addis Ababa, Ethiopia, 14–17 September 2015; pp. 1–6. [CrossRef]
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, Port Elizabeth, South Africa, 3–4 November 2014; pp. 95–107.
3. Blanco-Novoa, Ó.; Fernández-Caramés, T.M.; Fraga-Lamas, P.; Castedo, L. An Electricity Price-Aware Open-Source Smart Socket for the Internet of Energy. *Sensors* **2017**, *17*, 643. [CrossRef] [PubMed]
4. Vine, D.; Buys, L.; Morris, P. The Effectiveness of Energy Feedback for Conservation and Peak Demand: A Literature Review. *Open J. Energy Effic.* **2017**, *2*, 7–15. [CrossRef]
5. Wang, F.; Hu, L.; Zhou, J.; Zhao, K. A Data Processing Middleware Based on SOA for the Internet of Things. *J. Sens.* **2015**, *2015*, 827045. [CrossRef]
6. Khana, R.; Sethi, P.; Sarangi, S.R. Internet of Things: Architectures, Protocols, and Applications. *JECE* **2017**, *2017*, 9324035. [CrossRef]
7. Lin, H.; Bergmann, N.W. IoT Privacy and Security Challenges for Smart Home Environments. *Information* **2016**, *7*, 44. [CrossRef]
8. Beligianni, F.; Alamaniotis, M.; Fevgas, A.; Tsompanopoulou, P.; Bozanis, P.; Tsoukalas, L. An internet of things architecture for preserving privacy of energy consumption. In Proceedings of the Mediterranean Conference on Power Generation, Transmission, Distribution and Energy Conversion (MedPower 2016), Belgrade, Serbia, 6–9 November 2016; pp. 1–7. [CrossRef]
9. Stojkoska, B.R.; Trivodaliev, K.V. A review of Internet of Things for smart home: Challenges and solutions. *J. Clean. Prod.* **2017**, *140*, 1454–1464. [CrossRef]
10. Khatu, M.; Kaimal, N.; Jadhav, P.; Rizvi, S. Implementation of Internet of Things for Home Automation. *IJEERT* **2015**, *3*, 7–11.
11. Sagar, V.; Kusuma, S.M. Home Automation Using Internet of Things. *IRJET* **2015**, *2*, 56–72.
12. Lee, C.-H.; Lai, Y.H. Design and Implementation of a Universal Smart Energy Management Gateway based on the Internet of Things Platform. In Proceedings of the 2016 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 7–11 January 2016; pp. 67–68.
13. Al Faruque, M.A.; Vatanparvar, K. Energy Management-as-a-Service Over Fog Computing Platform. *IEEE Internet Things J.* **2016**, *3*, 161–169. [CrossRef]
14. Li, X.; Nie, L.; Chen, S.; Zhan, D.; Xu, X. An IoT Service Framework for Smart Home: Case Study on HEM. In Proceedings of the IEEE International Conference on Mobile Services, New York City, NY, USA, 27 June–2 July 2015; pp. 438–445. [CrossRef]
15. Viswanath, S.K.; Yuen, C.; Tushar, W.; Li, W.-T.; Wen, C.-K.; Hu, K.; Chen, C.; Liu, X. System design of the internet of things for residential smart grid. *IEEE Wirel. Commun.* **2016**, *23*, 90–98. [CrossRef]
16. Rasheed, M.B.; Javaid, N.; Ahmad, A.; Awais, M.; Khan, Z.A.; Qasim, U.; Alrajeh, N. Priority and delay constrained demand side management in real-time price environment with renewable energy source. *Int. J. Energy Res.* **2016**, *40*, 2002–2021. [CrossRef]
17. Hussain, H.M.; Javaid, N.; Iqbal, S.; Hasan, Q.U.; Aurangzeb, K.; Alhussein, M. An Efficient Demand Side Management System with a New Optimized Home Energy Management Controller in Smart Grid. *Energies* **2018**, *11*, 190. [CrossRef]
18. Khan, A.; Javaid, N.; Yousafzai, A.A. A priority-induced demand side management system to mitigate rebound peaks using multiple knapsack. *J. Ambient. Intell. Humaniz. Comput.* **2019**, *10*, 1655–1678. [CrossRef]