Integration of Green aspect inside Internet of Things standard

Thierry Monteil

Integration of Green aspect inside Internet of Things standard

Thierry Monteil
IRIT, Université de Toulouse, INSA
Toulouse, France
thierry.monteil@irit.fr

Abstract— **The Internet of Things is seen as a solution for many problems such as helping to create smart cities, industry 4.0 or even smart grids. The objective is in particular to offer a better service, to automate a set of services or to save money, particularly on energy consumption. Nevertheless, the complexity of these systems of systems requires the use of standards to deploy and make interoperable all of those connected objects and needed services. The oneM2M standard aims to provide a service layer to create those complex IoT systems. There are many scenarios and deployments in the literature and in standardization documents showing how the use of IoT can save or better manage energy consumption in a particular area, as opposed to energy consumption by the IoT system itself is little studied. This aspect, particularly in the oneM2M standard, is currently not very present except for interfacing with the cellular network and considering the capabilities of 5G in terms of reducing energy consumption. On the other hand, at the level of services, data management or the architecture to be deployed to set up an IoT system ranging from the sensor to the cloud, everything remains to be done. In this article, we propose to raise awareness on the one hand of the actors of IoT standardization of the energy impact in the definition of standards and on the other hand of the developers of IoT stacks and applications. For this, we first develop an environment to measure the energy impact in the choice of the use of a standard and its implementations then in a second time, we identify a set of recommendations both on the evolutions possible of the standard but also on the choice of implementation and deployment of applications and IoT stacks.**

Keywords— Internet of things, standardization, self-energy consumption, REST architecture

I. INTRODUCTION

The Internet of Things is a field that continues to develop. The domains of application are very varied and initially were rather in a very vertical vision: logistics of goods, e-health, home automation, etc. This has evolved towards a more horizontal and multi-domain vision with for example: industry 4.0 or even the smart city. The main reasons are the greater economic potential and a certain maturity of the technologies. However, this amplified the complexity with systems of systems, interoperability needs and data management which had to rely on new concepts such as ontologies. This evolution has amplified the need for standardization not only on equipment and communication layers but also on services and their access to drive, interact, monitor these IoT systems.
This article will focus on a standard for the IoT service layer called oneM2M [1]. There are other initiatives to try to define a common layer of services but oneM2M is distinguished by the support of standardization bodies.

At the same time, the field of green IT is becoming increasingly important given the environmental impact of ICT and its strong growth. Standardization bodies have produced numerous documents. PKIs have been defined for different levels of a system in terms of production consumption [2], in communication systems [3], for end of life of equipment [4], etc.

The crossing between the standardization of the IoT and the green IT has for the moment been made almost exclusively in the fields of applications such as the smart city [5] or even the smart grid [6]. However, work does exist, for example, on considering the energy consumed by specific technology like communication network [7], type of architecture like edge [8] or an end-to-end IoT system [9]. Tools exist also to measure the energy impact in the ICT world but partially adapted to the IoT (greenspector for example for the WEBsmartphone [10]). Standardization bodies have not taken the sustainable aspect into account in defining the standard itself. However, this could be integrated into the services, into the mechanisms or even into the architecture that the standard offers. The increasing deployment of IoT systems will ultimately have an increasingly strong impact on the energy that they themselves consume.

In this work, an environment of test has been created to measure the impact of deployment of the oneM2M standard in term of energy. First, elements on the oneM2M standard will be given, then we will present the test platform which was used to deploy the tests and collect the energy measurements. Finally, we will give the results on 3 opensource IoT stacks and a first synthesis to suggest improvement of IoT standard in term of self-consuming energy management.

II. CONTEXT

A. oneM2M Principle

oneM2M is driven by a consortium of five standardization organizations bringing together more than 200 industrial partners and laboratories. This standard aims to define a set of services as well as resource trees that make it possible to create a topology of connected objects. oneM2M then makes it possible to provide identical access to a set of heterogeneous technologies and data.

The standard specifies several types of nodes, the main ones being:

- **Infrastructure Node (IN):** This node is able to centralize links to the different middle nodes and allowing high-level user applications to interact with all nodes. the IN is generally deployed on the cloud. It hosts the service layer called Common Service Entity (CSE).
- **Middle Node (MN):** This is a device containing sensor data. He is directly connected to IN servers and sensors. The MN generally acts as a gateway between the long-distance network and the local sensor/actuator networks. It hosts also a CSE.
- **Application Service Node (ASN):** This is a node deployed on objects with a good processing capacity and therefore having the possibility of also hosting the service layer of oneM2M
- **Application Dedicated Node (ADN):** it is a node hosting an application called Application Entity (AE) but not containing the oneM2M service layer.

A common operating principle is as follows:

1. The MN registers with the IN and everyone creates a link in their database (remote IN/MN server)
2. Sensors registers with the MN and creates an Application Entity (AE) containing the description application and sensor data.
3. An End User (a user application such as a smartphone application) may have access to data through a REST API, receive notifications when they are updated, etc. Most of access are done at the IN level.

B. oneM2M data management

The oneM2M service layer is based on REST architecture. Data and services are represented in the form of a resource tree. In oneM2M, this takes the form of a creation of a CSE (Common Service Entity) as the root of the tree. Resources can be of a multitude of types. Here are the main ones:

- **Application Entity (AE):** This is a resource that symbolizes a sensor, an actuator or any other connected object or an application. Inside this resource, we will store data and the actions that we can carry out
- **Containers (CNT):** they are under an AE resource and group together several other resources to store them.
- **Content Instance (CIN):** they are located in the containers and most often represent the value collected by sensors, or a description of possible actions on an actuator.
- **Subscriptions (SUB):** they are located mainly in the containers and define an action to be performed for exemple when a content instance is added in this same container. More generally, they make it possible to subscribe to any modification of the resource in which they were created by setting up a notification mechanism.

C. Three oneM2M opensource stacks

The oneM2M has several industrial and academic implementations with different features. Three of them are accessible as opensource:

- **ACME** is an opensource implementation of CSE created by Andreas Kraft [11]. A subset of oneM2M resources is proposed. The goal is to produce a version simple to install, configure, maintain, extend and use. The main use is for educational purpose. This IoT stack is written with python. The architecture of ACME is based on a python code to run. The data are stored in file through *Tinydb* and the access is made with an API. This choice of database is one of the reasons of limitation in education use case. The second limitation is due to time response of ACME but as mentioned the goal was not to have an optimized responsive IoT stack. ACME offers an user interface accessible with a browser (figure 1).

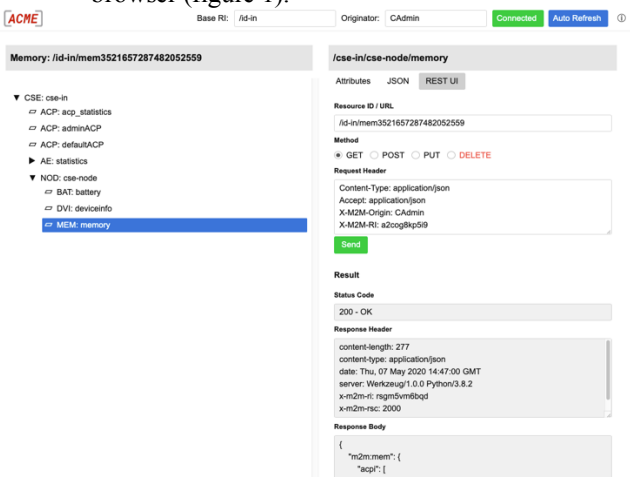

Fig. 1. ACME User interface

Developers have some information to help them to understand the structure of ACME in term of components (figure 2) and class hierarchy. Some guidelines are also offered to help to develop and integrate new applications in ACME.

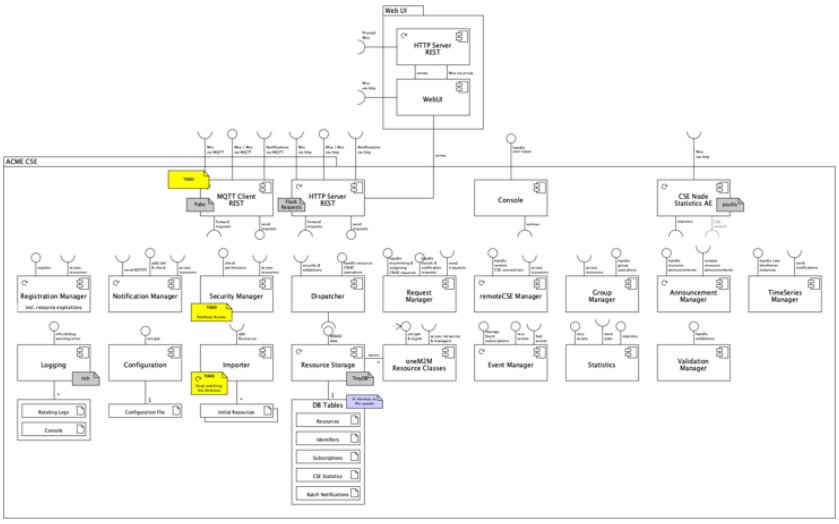

Fig. 2. ACME component architecture

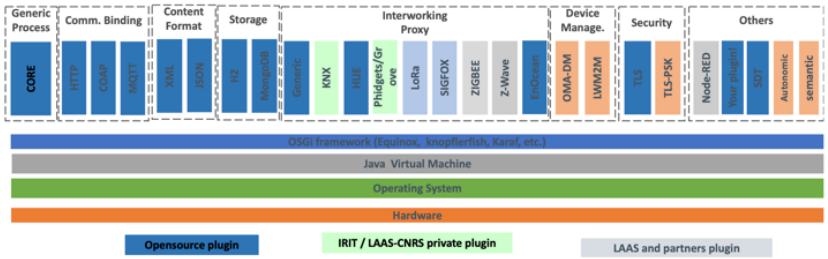

Fig. 3. OM2M architecture

- **Mobius** [12] is developed by KETI. It is an opensource implementation of CSE. It provides the service layer in term of: registration, data management, subscription and notification, security, etc. If also offers the concept on interworking for connection with different technologies. Mobius is certified by TTA for release 1 of oneM2M. It is written in node JS language. Mobius use an external MySQL data base to store data and MQTT protocol (figure 3). MySQL server and MQTT broker are needed.

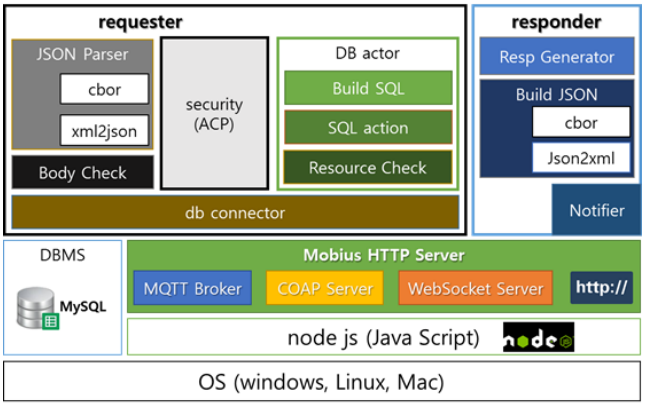

Fig. 4. Mobius architecture

- OM2M [13,14] has been created at LAAS-CNRS / IRIT. It has been selected by eclipse foundation as eclipse OM2M. It provides CSE based on release 2 of oneM2M. It is based on java and use a H2 database in memory or remotely. Documentation is available on eclipse website. Tutorials and MOOCs are also available. The architecture is based on plugins for all specific capabilities (figure 4). Installation is simple if you are using only bytes code, it is a bit more complicated with the java source due to dependency and version management of external packages.

Those three IoT stacks are interesting because they use different language: python, node JS, java. Their philosophy is different made for education, research or industry. Their architecture is also different (table 1). So, this subset of oneM2M implementation should have different behaviors in term of self-energy consumption.

TABLE I. COMPARISON OF THREE ONEM2M OPENSOURCE STACKS

	oneM2M release	language	documentation	installation	use	development	performance
ACME	R3	Python	Good on github	Very easy	Very easy	documented	limited
Mobius	Certified R1	Node JS	simple on github in English	not so simple on linux, need to cross different sources of information	very easy	no documentation in English	good
OM2M	R2	java	good on eclipse foundation	simple for bit code, complicated for source code	very easy	documented	good

III. EXPERIMENTATION

A. *Configuration of the testing platform*

The platform (figure 5) uses:

- an HP elite book 840 computer with an I5 Vpro quad-core processor and 8 GB of RAM. The battery has been removed so as not to disturb the power measurement. The operating system is Linux ubuntu 20.04.4. This computer will host an Infrastructure Node of each implementation to be tested.
- a macbook pro with an I7 quadcore processor and 16GB of RAM with *Mac OS monterey*. This machine will host the python client responsible for interacting with the IN. It will play the role of ADN in the oneM2M architecture. In addition, for the measure on the subscription/notification mechanism, it will also host an http server to receive the notifications sent by the IN.
- a wired ethernet network at 100 Mb/s is set up with an ASUS RTAC51U router. This network is isolated from any disturbance and the WIFI networks is deactivated.
- a rhodes&schwarz HMC 8015 power analyzer measures the power consumed by the machine hosting the IN. Measures are done every 0.1 seconds and stored in a file.

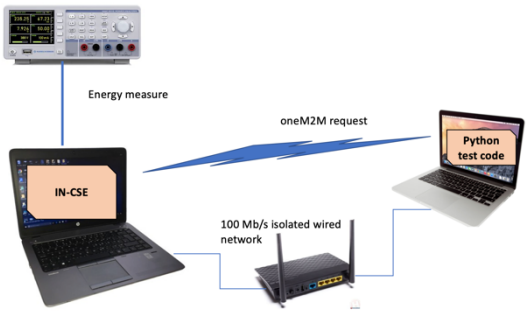

Fig. 5. Test platform with an Infrastructure Node

Most of the tests are run during a large period compared to the time of the oneM2M requests to integrate energy consumed due to cooling of the computer. Nevertheless, due to the high response time of ACME compared to Mobius and OM2M, we had to make different tests for ACME in term of number of resources manipulated. With ACME, we manipulate a burst of 100 resources and for Mobius and OM2M we manipulate burst of 1000 resources. By this way, we stay in the same interval of time for each experience.

In order to calibrate the energy needed by the computer to run the operating system and the basic services, a first set of measurements is made for 3min. This gives an estimation of the power required to power the machine which is equal to 9.34 W. The curve shows regular peaks (with a maximum of 23,25W) corresponding to linux service activation (figure 6).

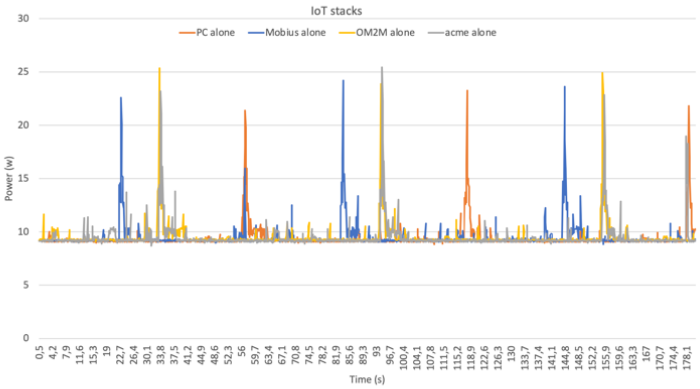

Fig. 6. Power consumption for computer and operating system

B. *Energy Consumption behavior*

Standalone stack: For each IoT stacks, we start the stack and waiting for stabilisation of energy consumption to remove the peak due to initialisation phase (figure 7). This value will help to calculate the over consumption due to creation and delete of oneM2M resources. To estimate the over consumption due to IoT stack we make the difference between computer without IoT stack and computer with the oneM2M stacks (figure 6). This consumption of IoT stack is due to periodic activities in IoT stack to get possible message from the network or activities on the database. ACME and Mobius have a consumption respectively of 3.50 and 3.11 kWh. OM2M has a greater consumption with 5.3 kWh. The reason is the Java virtual machine that creates an overload activity on processor.

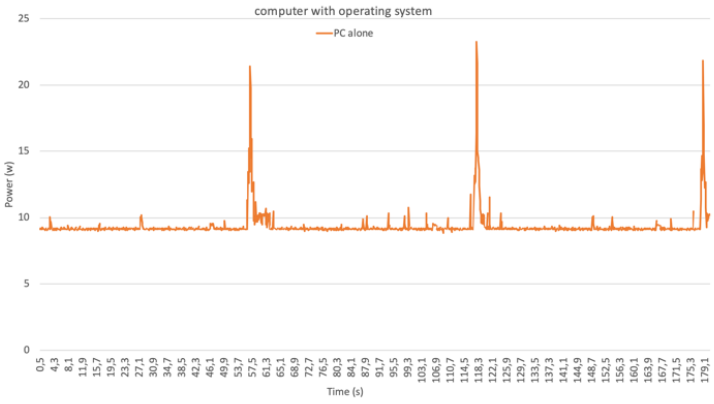

Fig. 7. Power consumption with the different IoT stacks with no request

Application Entity manipulation: In this test, a burst of Application Entity creation (1000 for Mobius and OM2M, 100 for ACME) is done any activity is stopped during 1min and then a burst of deletion of AE is done.

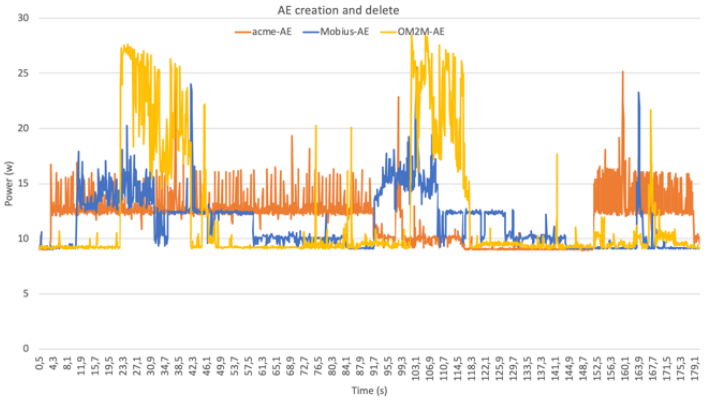

Fig. 8. Power consumption with the different IoT stacks for AE manipulation

The behaviors are very different (figure 8). OM2M consumes a lot of power during a short time with a maximum of 28,54 W. Mobius consumes less power but during a longer time and have different phases of consumption with a first phase with different peaks with a maximum of 25.59 W and after a phase with stable consumption around 12.2 W. For ACME the consumption is made with a consumption around 12.2 W and many small peaks with a maximum of 25.16W. We should also notice that for ACME the time to process the only 100 creation and delete of AE is longer then Mobius and OM2M to manage the 1000 requests of creation and deletion. We can estimate the power consumption for AE creation and delete for each task (Table 2). The creation of an AE cost a lot of power for ACME compared to the deletion. Mobius and OM2M are nearest with a power between 1.66 to 2.45 for creation and deletion of AE.

TABLE II. ESTIMATED POWER PER REQUEST FOR APPLICATION ENTITY

	Total power during burst (w)	Duration of burst (s)	Estimated power of IoT stack (w)	Estimated power per request (w)
Creation ACME	11354	88	3131.9	31.32
Creation Mobius	6312.42	48.1	1818.29	1.82
Creation OM2M	4245.4	19.2	2451.49	2.45
Delete ACME	3702.18	27.1	1170.14	11.7
Delete Mobius	6509.33	51.9	1660.15	1.66
Delete OM2M	3532	16.1	2027.72	2.02

CIN manipulation: In this test, a burst of content Instance (CIN) creation is made then we stop any activity and then a burst of delete of the CIN is done. The CIN resource is made to collect data from the system for example the set of value of a sensors. A new behavior for ACME could been seen for the delete with a energy consumption that depends on the number of CIN record in the database. The processing time of the algorithm in the database depend on the number of recorded in the database for CIN.

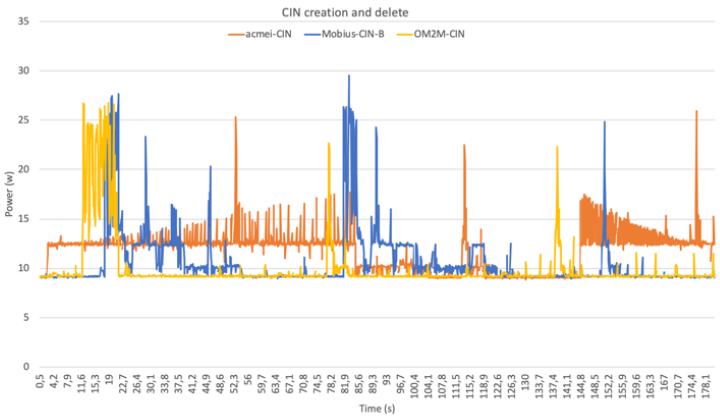

Fig. 9. Power consumption with the different IoT stacks for CIN manipulation

We can estimate the power consumption for CIN creation and deletion for each task (figure 9). All stacks included ACME have behavior in term of power consumption that look like the same then previous resources. The cache mechanism of in memory database of OM2M helps to have a very efficient delete of CIN with 0.12W per delete (table 3).

TABLE III. ESTIMATED POWER PER REQUEST FOR CONTENT INSTANCE

	Total power during burst (w)	Duration of burst (s)	Estimated power of IoT stack (w)	Estimated power per request (w)
Creation ACME	10443.44	82.2	2763.24	27.63
Creation Mobius	4539.56	36.4	1138.6	1.14
Creation OM2M	1929	9.6	1032.33	1.03
Delete ACME	4949.26	37	1492.24	14.92
Delete Mobius	4908.85	40.50	1124.81	1.12
Delete OM2M	653.88	5.7	121.31	0.12

LAST request: In this test, a creation of one content instance is done and then a burst to get the last content Instance (CIN) created then we stop any activity and then all resources are deleted. We can estimate the power consumption for creation and get of this value inside an application and delete for each task (table 4). This consumption is very low for all IoT stacks. This is important because one of the main goal of IoT stack is to expose the collected values. ACME is really efficient compared to the other resources management. This is due to the use of in memory simple data base. OM2M has a cache on database and this help also to get low consumption. Mobius is just over the two other stacks in term of consumption, this is due to the management of the remote database and processing activity created even after the response to the application test.

TABLE IV. ESTIMATED POWER PER REQUEST FOR GET LAST CONTET INSTANCE

	Total power during burst (w)	Duration of burst (s)	Estimated power of IoT stack (w)	Estimated power per request (w)
ACME	1493.71	11.5	419.23	0.42
Mobius	3866.93	28.5	1204.09	1.2
OM2M	1605.18	7.4	913.78	0.91

Subscription and notification mechanism with SUB resource: In this test, we make a burst of subscription (SUB) on a container that contents Content Instance resources (CIN), we stop any activity and then we create a new CIN in the container that create a burst of notification then we have a burst of deletion of the SUB (figure 10).

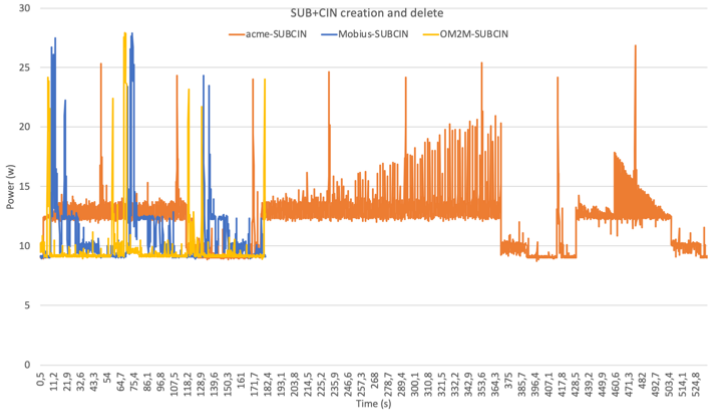

Fig. 10. Power consumption with the different IoT stacks for SUB creation, delete and notification mechanism

To estimate the power consumption for notification mechanism (table 5), we have estimated before the value for the subscription creation with a test with only requestion for subscription creation. A soustraction of this value help to estimate the notification cost in term of power consumption. That is to say the cost to get the information about the remote http server and the sending of the notification to this http server. ACME still have a very high power cost. Mobius and OM2M have a very low cost compared.

TABLE V. ESTIMATED POWER FOR NOTIFICATION MECHANISM

	Total power during burst (w)	Duration of burst (s)	Estimated power of IoT stack (w)	Estimated power per request CIN+notification (w)	Estimated power per notification (w)
ACME	26346.47	198.9	7762.63	77.6	49.99
Mobius	5477.11	44	1366.05	1.37	0.23
OM2M	2247.5	11.5	1173.02	1.17	0.14

Multi-level Architecture with an Infrastructure and Middle Nodes: the goal is to measure the impact of use of two CSEs based on one Infrastructure node and one Middle node. Application requests are sent to the IN node and a redirection to MN node is done by the oneM2M stack. The two CSEs are run on the same computer to be able to measure the processing power consumption (figure 11).

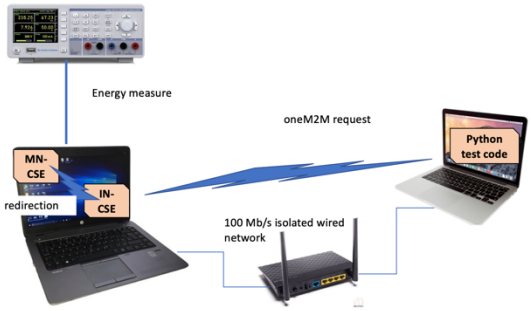

Fig. 11. Deployment of one IN and one MN node with OM2M stack

This architecture is not really realistic because IN nodes are deployed on cloud and MN nodes on gateways and devices in real application. This creates also a communication over wide area network. Nevertheless, do real measures on such architecture is not trivial and most of the time simulation replaces real measures.

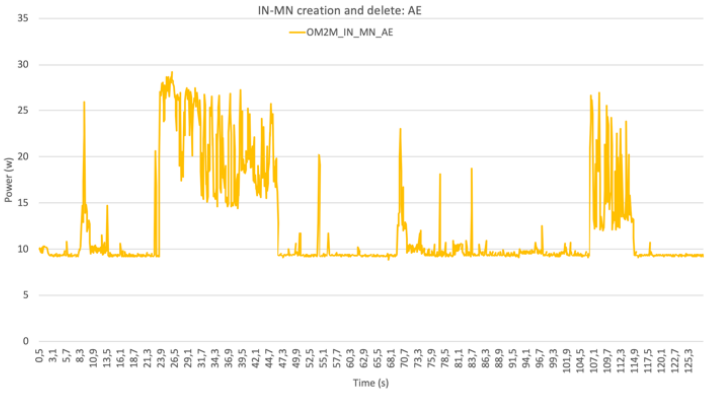

Fig. 12. Power consumption with on IN and one MN node with OM2M stack

The measures are shown on figure 12. By using the value measured for AE creation with OM2M we can deduce the over cost of redirection of the request from IN CSE to MN CSE: 0,39W (table 6). It over costs around 15% to use IN and MN node and in real world we should add the network impact. So the use of redirection mechanism has an significant impact on energy consumption.

TABLE VI. ESTIMATED POWER FOR REDIRECTION MECHANISM

	Total power during burst (w)	Duration of burst (s)	Estimated power of IoT stack (w)	Estimated power per request (w)	Overcost of redirection (w)
OM2M	5061.76	23.8	2838.05	2.84	0.39

IV. LESSONS FROM EXPERIMENTATION

The results of a set of experiments allowing to better understand the energy impact of the implementation of the oneM2M

standard. We were also able to study the behavior in terms of energy consumption of the manipulation of a subset of the resources described in the standard. The question is how can this impact the oneM2M standardization community: standard contributors, stack developers and oneM2M users.

A. Analysis

Several aspects could be analyzed from the experiments done:

- **programming language:** The 3 IoT stacks have been chosen because of their opensource characteristic. It is allowing to test 3 stacks developed with different programming languages. ACME is written in python, Mobius in node JS and OM2M in java. These language choices were clearly guided by a desire for easier portability on computer and operating systems. The choice of these languages is also to a lesser extent derived from the need to have a strong connection with the concepts of the Internet by the very nature of the IoT. Node JS and java are two languages strongly coupled to the Internet, python to a lesser extent. This choice was made to the detriment on the one hand of the processing power compared to a language compiled in binary which can be directly used on the processors and on the other hand of the energy necessary to execute them. The 3 implementations do not have a big difference in consumption when they are not requested, on the other hand as soon as requests is processed the difference between ACME in python and the 2 other implementations is very important with an average (by removing the 2 extreme cases) of the order of 20 times more energy consuming. The choice of language alone cannot explain this difference. The choice of software design also had an influence. ACME would like to be use for education and therefore easy to understand but sometime not optimized in term of performance.
- **database:** Storing the resource tree necessary to guarantee the stateless aspect of REST requests requires having an efficient information storage mechanism. Naturally, IoT stacks are using a database. The diversity of databases: objects database, SQL, NoSQL, etc. and the choice made by implementations has a considerable impact on performance and energy consumption. It is very likely that this is one of the bottlenecks of ACME. The cache mechanisms that can be activated on certain databases can reduce the energy consumed, especially when the resources handled are either often the same, or the most recently produced, or rather have read access. ACME also shows specific behavior in particular on the destruction of resources. The time and therefore the energy are strongly impacted by the number of resources in the database. IoT stacks must therefore be vigilant about minimizing the impact of the number of records in a database and the query time for the main used by oneM2M specific application.
- **Distributed or not:** the distribution of the functionalities necessary for the IoT stack has an impact on energy. We see the impact of the database being distributed rather than embedded directly in the process memory of the IoT stack. This is the case for example of MOBIUS which tends to consume energy longer. Similarly, if we had put a request dispatcher feeding a distributed service architecture on a cloud, we could have measured the energy impact of using more computer and more processes.
- **IN/MN/ASN:** oneM2M build complex architecture based on IN, MN and ASN CSE for the service layer part. However, we found that adding an IN+MN had an impact on power consumption. Due on the one hand to the creation of several processes on the computers but also to the additional mechanisms to be put in place such as the redirection of requests.

B. Proposals

Derivated form the tests and analysis, a list of proposals and recommandations are made:

- **programming language:** Choose an efficient programming language to develop the IoT stack
- **Conception and development of IoT stack:** Think about performance and energy consumption when you develop an IoT stack
- **Database:** Choose a database and optimization parameters of database related to type of usage and requests made by applications (impact of distribution, cache, size of database, etc)
- **oneM2M node and energy:** The deployed architecture should be as simple as possible in term of number of nodes and processes created in computer.
- **Energy Monitoring:** resources to keep information on the energy consumed by both the computer, the CSE or the manipulation of the resources is necessary. This may be possible by using the energy measurements that a computer can make by itself and in particular on its processor over time. An energy resource can be attached to any oneM2M resources and give information on energy consumption when an access is made for creation, read, write or update. This value will be static or evaluated by the CSE. Making those resources available to users will promote consideration of energy in the use of oneM2M.
- **QoS level:** The response time for oneM2M request is one the Quality of Service indicator. From now, IoT stack try to respond to a request as soon as possible. oneM2M service layer should introduce level of response time for specific deployment. With low level of response time for non-critical application, we can adjust energy consumption for a request. For example, IoT stack can reduce the frequency of processor based on DVFS mechanism [15] or stop core or processor in a computer. This implies to add a negotiation mechanism inside oneM2M CSE and inside oneM2M request. This can be done at the level of CSE resource or when an AE is created. This implies new parameters

in oneM2M request or new resources dedicated to describe QoS and energy impact possibility.

- **Energy budget:** Energy for the IoT device may be critical for unplugged equipment's. For those equipment battery, solar panel or any local energy source should be used. The connection between energy source, hardware and oneM2M CSE should be done to manage energy. One way to do that is to manage an energy budget for oneM2M request. The global CSE can evaluate its energy budget at a specific time. This can be improved with prediction of energy production to get prediction of energy budget. The CSE must also have the capacity to measure by itself the energy consumed with various accesses to oneM2M resources. Then when an application makes a request either the CSE can evaluate that the energy consumption of this request will be too high given its current energy budget and, in this case, returns an error code or the request can itself express the quantity amount of energy it's willing to pay to be answered. The idea here is to bring a market mechanism for energy at the level of requests for CSEs. This also implies to add new resource to describe energy budget and parameters on request to manage exchange between applications and CSEs.

V. Conclusion and perspective

In this article, a physical platform of test has been created that used the same application test program to estimate power consumption of the 3 IoT stacks: ACME, Mobius and OM2M. The programming languages used in the 3 stacks are different. The architecture of the software is very different. Cumulated with the behavior of the different oneM2M resources management made by stacks implies different power consumption profile.

Energy consumption of IoT stack should be taken in account in the future. This should be made at the level of standardization organism, developers of oneM2M IoT stacks, developers of IoT applications and finally at the level of deployment of IoT solutions.

Based on those recommendation, we have started to work on new IoT stack called lightom2m [16] based on C++ language. Low impact on memory and processing are the first goal of this new work. The second goal will be to be able to test and experiment low energy recommendation made in this paper.

Acknowledgment

This work has been done under the founding of European project StandICT.eu.

References

[1] http://www.onem2m.org

[2] Operational energy Efficiency for Users (OEU); Global KPIs for ICT Sites, ICT ETSI GS OEU 001.

[3] Access, Terminals, Transmission and Multiplexing (ATTM); Energy management; Operational infrastructures; Global KPIs; ETSI EN 305 200-3-1.

[4] Access, Terminals, Transmission and Multiplexing (ATTM); Broadband Deployment and Lifecycle Resource Management, ETSI EN 305 174-8.

[5] Adaptation of oneM2M for Smart City, oneM2M TR-0036- V0.3.0.

[6] oneM2M Use Case collection, ETSI TR 118 501.

[7] J. Finnegan, An Analysis of the Energy Consumption of LPWA-based IoT Devices, IEEE International Symposium on Networks, Computers and Communications (ISNCC), Rome, Italy, June 2018, DOI:10.1109/ISNCC.2018.8531068

[8] J. Mocnej, M. Miškuf. P. Papcun, I. Zolotová, Impact of Edge Computing Paradigm on Energy Consumption in IoT, IFAC 2018 Intenrational Federation of Automatic Control.

[9] L. Guegan, A.C. Orgerie. Estimating the end-to-end energy consumption of low-bandwidth IoT applications for WiFi devices. CloudCom 2019 - 11th IEEE International Conference on Cloud Computing Technology and Science, Dec 2019, Sydney, Australia. Hal-02352637.

[10] https://greenspector.com/en/home/

[11] https://github.com/ankraft/ACME-oneM2M-CSE

[12] https://github.com/IoTKETI/Mobius

[13] https://www.eclipse.org/om2m

[14] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, K. Drira, OM2M: Extensible ETSI-compliant M2M Service Platform with Self-configuration Capability, Procedia Computer Science, Volume 32, 2014, Pages 1079-1086, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2014.05.536.

[15] T. Kolpe, A. Zhai, S. Sapatnekar, Enabling improved power management in multicore processors through clustered dvfs, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2011, pp. 1–6.

[16] https://gitlab.irit.fr/sepia-pub/lightom2m