Smart IoT Sensor With LwM2M Management

University of Applied Sciences
Western Switzerland
SI Infotronics

SmartGrid

Darko Petrovic
Alexandra Andersson
Pascal Sartoretti

Contents

Introduction

This document describes the realization of a self-sustained IPv6 sensing device for the Internet-of-Things running with a rechargeable battery and harvesting energy from a solar panel. The first chapter gives an overview of the hardware with the main components of the device and their characteristics. Then the document focus on the energy consumption while operating in the network and the optimizations made throughout the protocol stack to save as much as possible the energy of the battery. We introduce equally our Web Interface for the management of the devices within the network. Finally, the current consumption of the sleepy device and the battery recharge are analyzed.

Hardware

This chapter gives an overview of main components of the device.

Bloc diagram

The bloc diagram of the smart sensor described in this document is shown in Figure 1. With the objective to design a self-sustained sensor, the device embeds a harvesting system to extract the energy from an external power source. In our case we decided to use a solar panel as a power transceiver element because the light is the most abundant energy available for such purpose. Connected to the solar panel, the power management unit (BQ25504) harvests the solar energy with an efficiency of 80%, recharges the battery and provides power to the whole system. A load switch after the PMU allows to efficiently startup the device while at low voltage. Another load switch between the USB and the PMU disables the sampling of the input voltage when the USB is used otherwise the USB power is cut every 16 seconds by the MPPT.

The device embeds the following sensors: temperature, humidity, pressure, illuminance, air quality, motion detector and microphone. A voltage and current measurement chip is incorporated to measure the solar input power and the battery voltage. With the shunt resistor of 10Ω after the solar input we can measure the current with a resolution 4µA up to 16.38mA. Except for the motion detector and the microphone, all sensors communicate through an I^2C interface. Thanks to an I^2C switch, the power of each sensor can be controlled directly from the SoC device (CC2538) reducing the current to its minimum while the system is in deep sleep mode. Finally an USB connection is available through which the battery can be recharged very quickly and configure the device for the first usage via a serial interface.

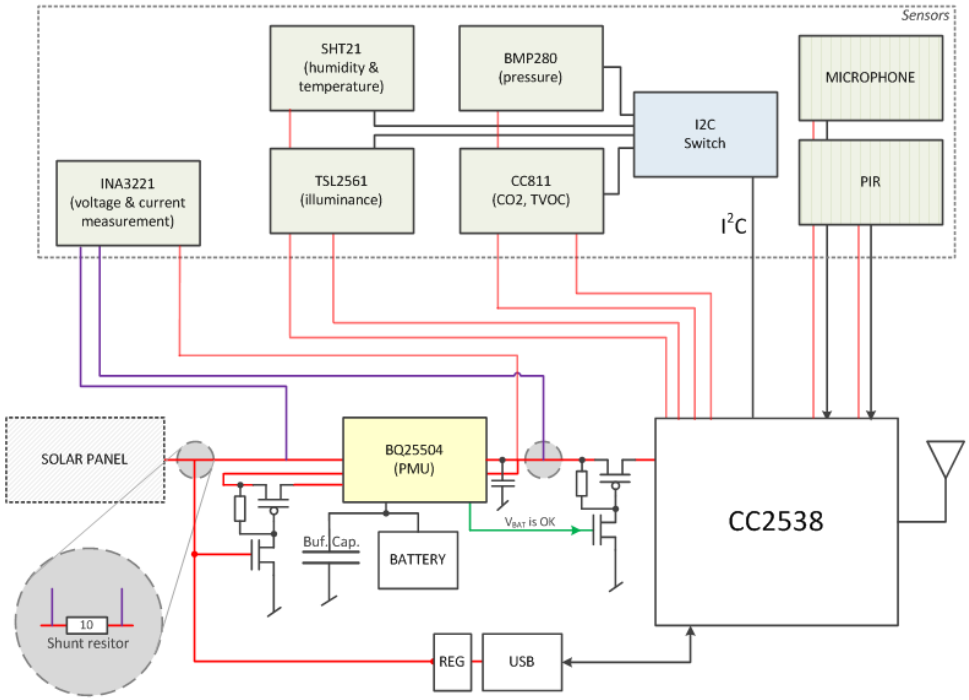

Figure 1.Bloc diagram of the device.

Components with their current consumption

The Table 1 shows the measured current consumption in active and sleep states for each component on the sensor. Thanks to the I^2C switch, we can save about 25 µA of constant current consumption while in sleep state. The current reported for the microphone and the motion detector take in account the whole circuitry around chip.

Name	Description	Current	
		Active	Sleep
CC2538	SoC device	10mA	1.3µA
BQ25504	Boost Converter	-	0.4µA
INA3221	Power measurement	326µA	0.5µA
SHT21	Temp. & Humidity	280µA	0.15µA
TSL2561	Illuminance	206µA	3.2µA
BMP280	Pressure	558µA	2.8µA
CCS811	Air quality	20mA	19µA
ICS-40310	Microphone	122µA (full circuit)	-
LHI968	Motion detector	1.5µA (full circuit)	-

Table 1. Current consumption of each component on the device.

The current consumption of the system while in deep sleep mode is only 4.9 µA. Due to the high active current of the air quality sensor and the microphone, these sensors are not intended to be activated for a device running on the battery alone. The USB cable must be plugged while using these sensors for a long period.

Battery

The battery used is the VL-2330, a Vanadium Pentoxide Lithium coin battery, with 23mm in diameter, a rated voltage of 3.0V and 50mAh of nominal capacity. The discharge characteristics and lifetime are shown in Figure 2. The battery can last about 50 days with a current consumption of 50µA. The voltages thresholds configured in the PMU for the battery protection are shown in Table 2. The VBAT_OK_PROG is set very low otherwise the PMU can remove the power from the system due to the voltage drop when performing a radio transmission.

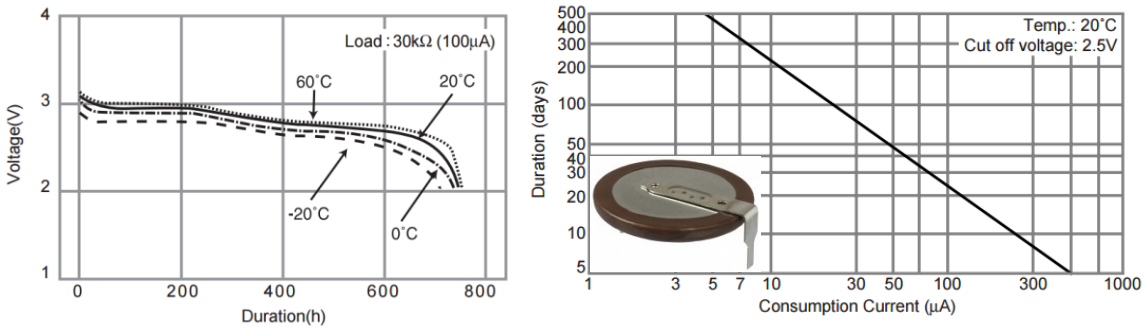

Figure 2. The battery with its discharge characteristics and lifetime duration.

Voltage	Values
VREF_SAMP	80%
VBAT_OV	3.42V
VBAT_OK_PROG	2.12V
VBAT_OK_HYST	2.43V
VBAT_UV	2.18V

Table 2. Voltages threshold configured with the resistors network of the PMU.

Solar panel

The solar panel SLMD121H04L (Figure 3) is made of 4 single cells of monocrystalline in series and provides an efficiency of 22%. The characteristics of the solar panel are shown in Figure 4. This solar panel was chosen because the results of our preliminary study (Figure 5) shows that it can provides 40µW of power in low light condition (250 lux) which is sufficient to sustain the power consumption of the device while in deep sleep and 200µW at 1000 lux to sustain the average power consumption of the system while performing measures.

Figure 3. IXYS SLMD121H04L with 4 single cells in series (43 x 14mm).

Symbol	Cell Parameter	Typical Ratings *)	Units
V_{oc}	open circuit voltage	2.52	V
I_{sc}	short circuit current	50.0	mA
V_{mpp}	voltage at max. power point	2.00	V
I_{mpp}	current at max. power point	44.6	mA
P_{mpp}	maximum peak power	89.2	mW
FF	fill factor	> 70	%
η	solar cell efficiency	22	%
$\Delta V_{oc}/\Delta T$	open circuit voltage temp. coefficient	-2.1	mV/K
$\Delta J_{sc}/\Delta T$	short circuit current temp. coefficient	0.12	mA/(cm²K)

*) All values measured at Standard Condition: 1 sun (= 1000 W/m²), Air Mass 1.5, 25°C

Figure 4. Solar panel electrical characteristics.

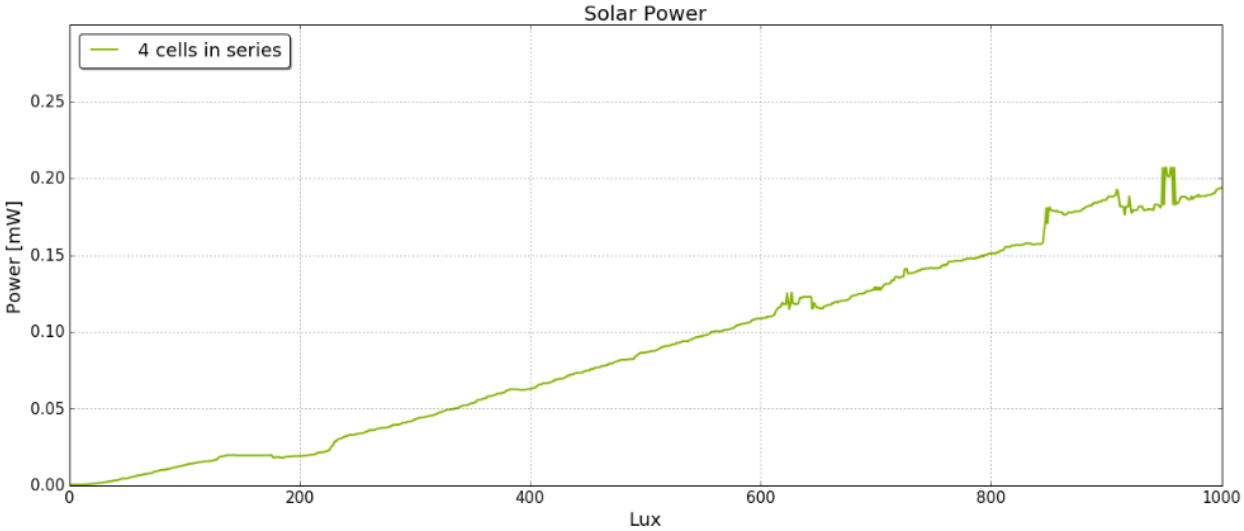

Figure 5. Power generated by the SLMD121H04L solar panel in low to middle light condition.

The board

The device and its housing are shown in Figure 6. The board has two buttons (user and reset) and two LEDs (yellow and red). The apertures on the case are made for the PIR sensor, the LEDs, the reset button, the microphone, the light sensor and the USB connector. The size of the board is 57x40mm. The case is realized with a 3D printer.

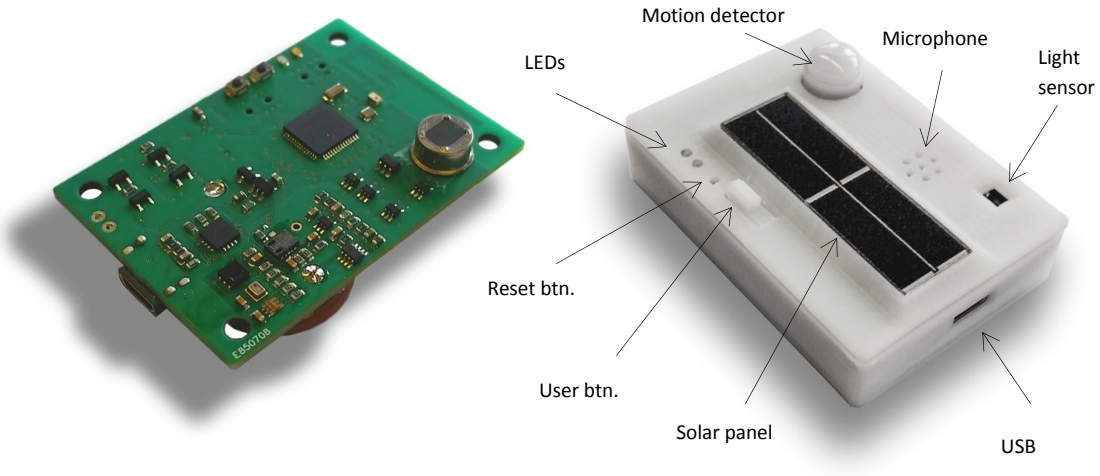

Figure 6. The device with its housing created with 3D printing.

Software

The sensor is running Contiki-OS v3.0, a free and open-source operating system for the Internet-Of-Things. The core of Contiki-OS provides the IP communication through a complete network protocol stack. The rest of the system is implemented as application libraries that are optionally linked with the program. The network protocol stack we're using in our project is illustrated in Figure 7.

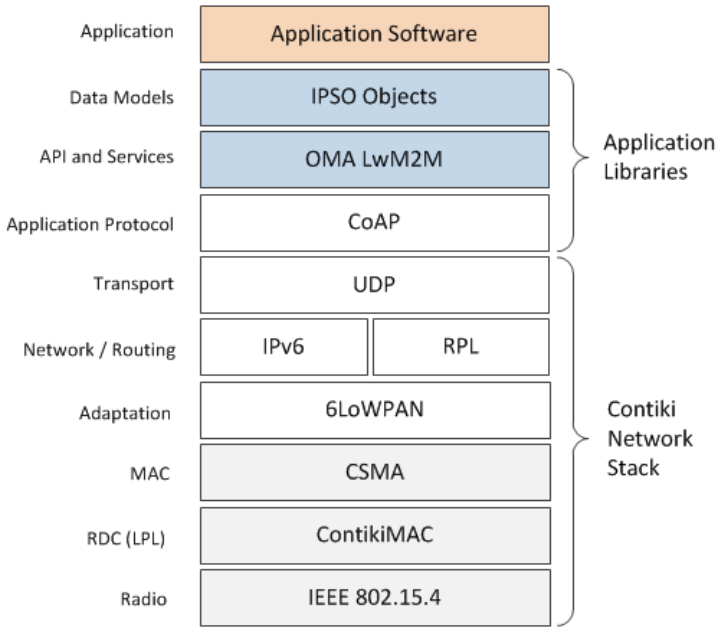

Figure 7. Network protocol stack covering all traditional OSI layers.

The first main objective in our project was to enable the sleepy device behavior for the node which joins the network as simple host and not as router. To achieve the best energy performance for the host behavior, some changes had to be made on the network protocol stack. The first one is the implementation of the Optimized Neighbor Discovery Protocol (RFC6775) [1] which introduces the concept of sleeping devices for IP communications. In [2] the protocol is evaluated in Cooja simulator against its counterpart Neighbor Discovery Protocol (NDP) in term of RS/RA messages exchange. It has been conclude that the protocol reduces the number of RS/RA exchange ratio messages in the network by 80%. We shows in this document the performances in term of energy consumption a device could benefit from this optimized protocol by considering both RS/RA and NS/NA messages exchange. We've furthermore enabled the mesh networking by activating the RPL protocol within Contiki and made some adjustments to the implementation to work together with the Optimized NDP.

Finally, to take full advantage of the Optimized NDP, we've implemented a smart RDC mechanism which deactivates the permanent idle listening (ContikiMAC) for the host when not required. Many researches have been carried out to improve the energy waste of the idle listening by implementing efficient MAC protocol but these researches are mainly focusing on the routing problem for the router behavior [3–5]. We don't address this problem in this document and keep the RDC activated all the time for the router which therefore require a larger battery or to be powered all the time by an USB cable. For the host the optimization of the idle listening is straightforward because it doesn't take part in the routing and every communication is initiated by itself.

The Low Power Listening and the Smart RDC

The Low Power Listening (LPL) is a common technique in Wireless Sensor Network for reducing energy consumption where nodes periodically wakes-up from the low power mode to sample the wireless channel and detect radio activity with the Clear Channel Assessment (CCA) mechanism. The actual implementation of the LPL mechanism in Contiki-OS turns on the RDC when the device boots and it is never turned off.

Justification to deactivate the LPL for a sleeping node

When activated all the time, this periodic wake-up of the radio is a waste of energy especially for nodes which doesn't take part in the routing within a mesh network. Furthermore, with the implementation of the optimized neighbor discovery protocol which introduces the concept of sleeping device, the deactivation of the LPL is more justified. The average current consumption when the LPL is activated and when the device is in deep sleep was measured and reported in Figure 8.

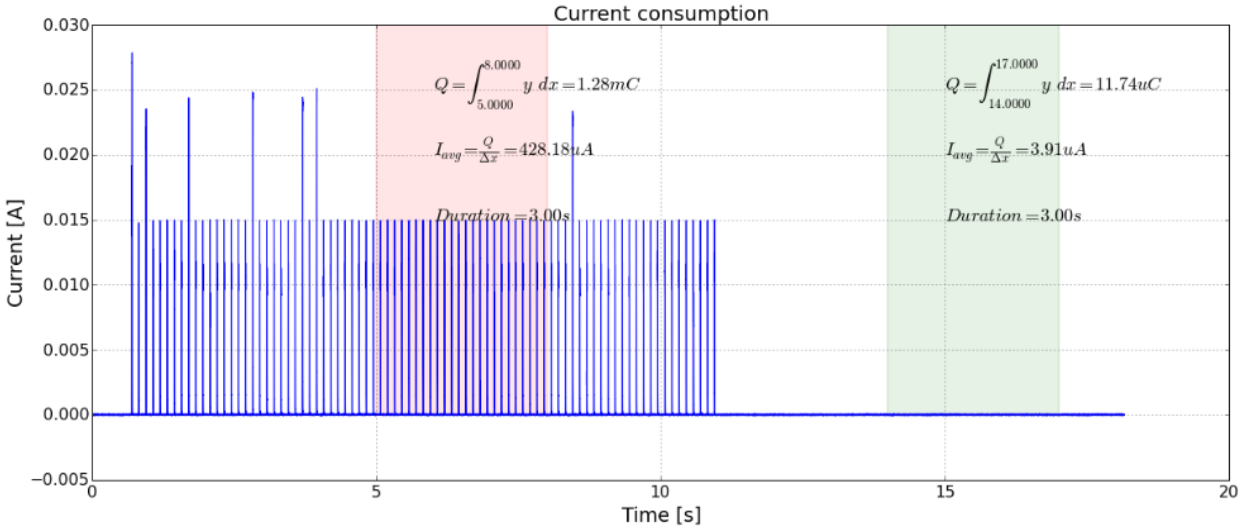

Figure 8. When activated, the LPL consumes 428uA on average whereas in deep sleep mode the average current is only 4uA.

Note that the deep sleep current when calculated by integrating the curve as in Figure 8 can in fact vary from 4 to 6 µA. The noise introduced by the measurement tool (Digitizer L4532A) with an external board make the precise measurement of very low current very hard. Using the Device Current Waveform Analyzer CX3322A, the deep sleep current was measured precisely to 4.9 µA and to 6.4 µA when the motion detector is activated.

We compute theoretically what is the excepted lifetime of the device while the LPL is activated. With a battery of 50mAh the expected lifetime is only 5 days and more than 1 year with 2xAA batteries (Table 3). With the actual battery capacity a device is not intended to work as a router without an USB cable plugged in.

Battery	Capacity [mAh]	Lifetime	
		LPL Activated	*LPL Deactivated*
2x AA	5000	484 days	142 years
CR2032	225	22 days	6 years
VL-2330	50	5 days	1.4 years

Table 3. Expected lifetime of the device with different batteries.

The code source of Contiki-OS was slightly modified to support the deactivation of the ContikiMAC RDC. For a sleeping device, the LPL is required only when the node is expecting a response from a request. Table 4 lists the type of messages which are expecting a response. Every time the node is making a request of the type listed in Table 4, the LPL is activated for 3 seconds. If no response is received during these 3 seconds the LPL is prolonged for 1 second until the response is received or the number of maximum attempt is reached.

Protocol	Message	Expected response
NDP	RS	RA
NDP	NS	NA
RPL	DIS	DIO
RPL	DAO	DAO ACK
CoAP	CON	ACK
CoAP	2.05 Content (M=1)	GET

Table 4. Type of message with the expecting response when the LPL must be activated.

The smart RDC was tested with the Cooja simulation platform. The simulator provides a view of the radio activity for each node in the simulation where we can verify that the RDC is correctly disabled. In Figure 9 four nodes are simulated. Node 1 is the slip-radio connected to a border router, node 2 is the router and nodes 3 and 4 are the hosts. Figure 10 shows the radio activity of these four nodes when the simulation is started. We can notice that the host nodes deactivate the RDC when they are not expecting a packet.

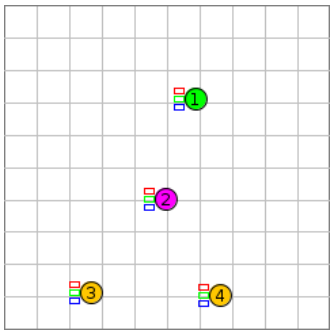

Figure 9. Simulation of 4 nodes in Cooja:
1) Slip-radio connected to the border router 2) Router 3,4) Host

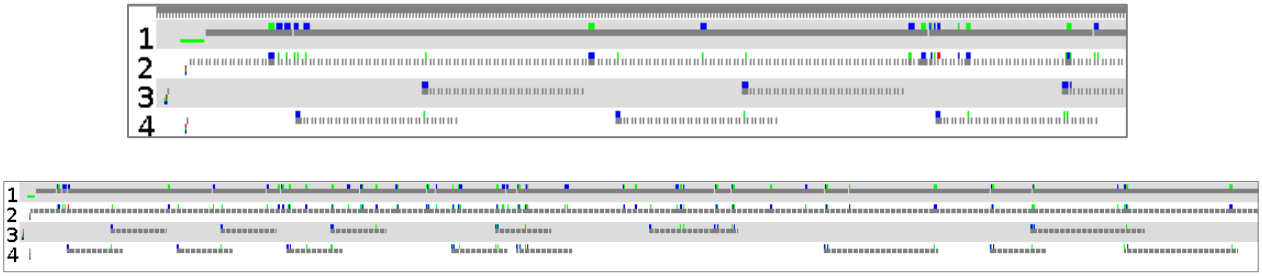

Figure 10. Radio activity of the nodes in Cooja.

The optimized Neighbor Discovery Protocol

The optimized ND process [1] provides support for sleeping host by making the interaction between the host and router a host initiated interactions and thus avoids multicast flooding of message and improves the interaction between sleeping host and routers. The IPv6 neighbor discovery in its basic form specified in RFC 4861 [6] was not designed for asymmetric reachability between the nodes in the network: a sleeping node is by definition not reachable at any time which makes the protocol inefficient.

The basic concept of the optimized NDP is that a host (6LN) registers itself to a router (6LR) for a certain duration during which the host can go to sleep. During this registration period the host doesn't perform Neighbor Unreachability Detection (NUD) to actively keep track of neighbor's changes. With the optimized NDP, a host speaks only to a router and uses the registration mechanism to keep the connection with this later. A device configured as a router registers itself equally as the host does and update its registration to its parent router too (Figure 11). The information provided by the border router (6LBR) is transmitted by the router to each device which registers to that router. The information from the 6LBR is then propagated to the entire network.

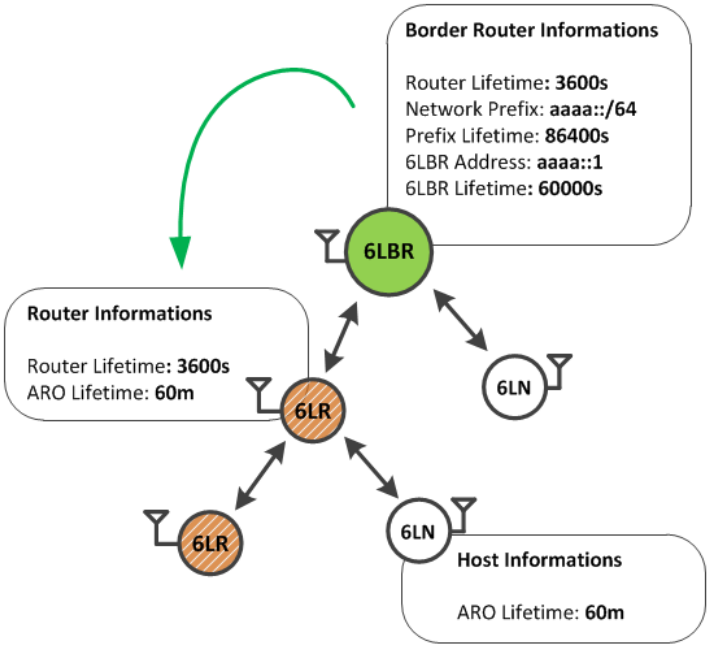

Figure 11. In the optimized NDP, routers and hosts register to its parent router with the new Address Registration Option (ARO) for a certain duration.

The optimized NDP exchanges the same type of messages as its counterpart NDP but with two new options: the Authoritative Border Router Option (ABRO) and the Address Registration Option (ARO).

When joining the network, the host or the router first sends a multicast Router Solicitation (RS) in order to find a parent router. A nearby router responds with a unicast Router Advertisement (RA) containing among others the new ABRO option. This new option contains the information relatives to the border-router which are its IP address, the version and its lifetime. Then the node send a Neighbor Solicitation (NS) containing the ARO option with the registration lifetime during which it'll be registered to the router. The router responds to the node with a Neighbor Advertisement (NA) containing the result of the registration. Note that there is two other messages not mentioned here which are introduced in the optimized NDP. The Duplicate Address Detection (DAD) and Duplicate Address Confirmation (DAC) messages. Both of these messages are used by the router to verify with the border-router, before accepting a registration, that the address trying to register is not already used by another node in the network. These messages are omitted here since we are using EUI-64 addresses which are supposed to be unique.

Once the registration is a success, the node send periodically a new registration with an NS just before the registration expires and an RS when the router, prefix or border-router lifetime is about to expires if not updated meanwhile.

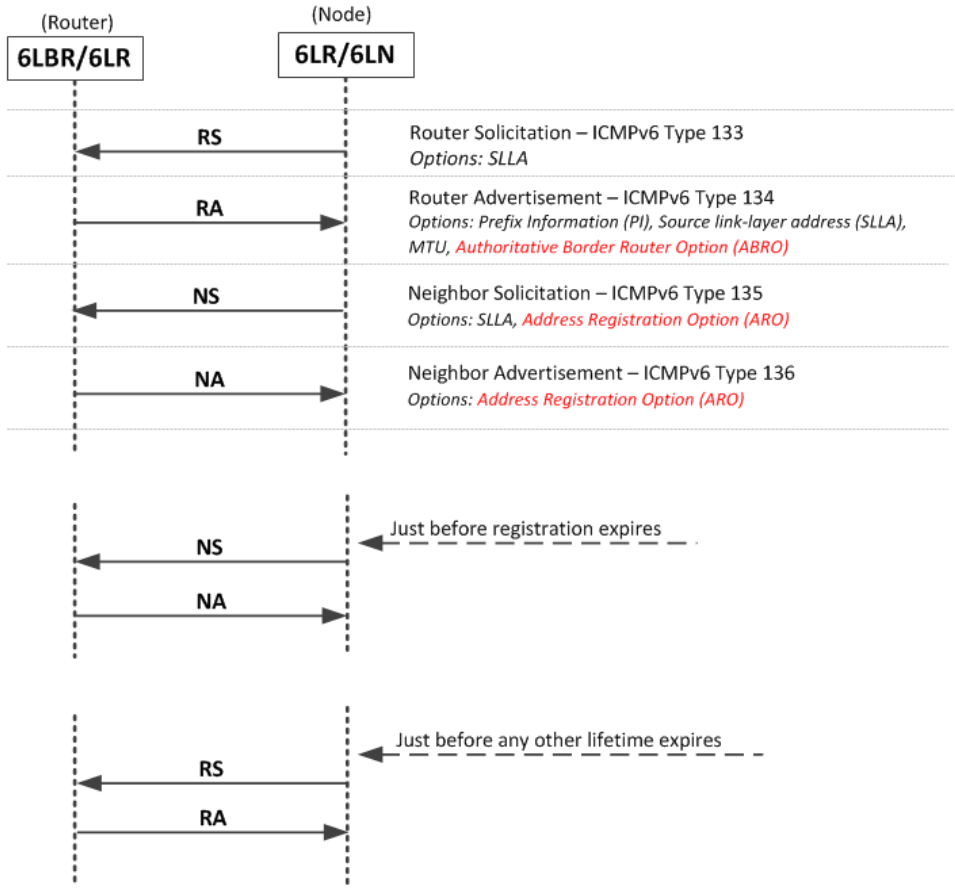

Figure 12. Messages exchange for the optimized NDP.

In the next chapter we'll evaluate the waste of energy of the standard neighbor discovery protocol against the optimized version for our device running on the CC2538 SoC, an ARM Cortex-M3 working at 16MHz.

Energy evaluation of the standard NDP in Contiki

In this section we evaluate the energy spent by the protocol while exchanging the periodic messages. We measure the current consumed by the device during the process and compute the total charge consumed for a full day of activity.

In the standard NDP a router send unsolicited multicast RA message periodically or when solicited by a RS message. With ContikiMAC, a multicast message is repeated during the whole RDC period so that every node receives the message. The periodicity of the RA message is not fixed by the protocol but can vary from 3 to 1800 seconds with a default value of 600 seconds in Contiki-OS. The exchange of the NS/NA messages is "periodic" too assuming that there is a periodic communication between the nodes. In fact, a neighbor's interface is REACHABLE for a specific duration after which the interface goes in the STALE state. When a node communicates to another node with the interface in the STALE state, the NUD verify the reachability of this

interface by exchanging NS/NA message. The reachable time of an interface varies from 5 to 15 minutes and it is determined when the interface is enabled. Since they are attached to an interface, the number of exchange must be multiplied by the number of interface.

Figure 13 and Figure 14 show the current consumption of our device while sending multicast RS/RA message and performing NUD with NS/NA message respectively.

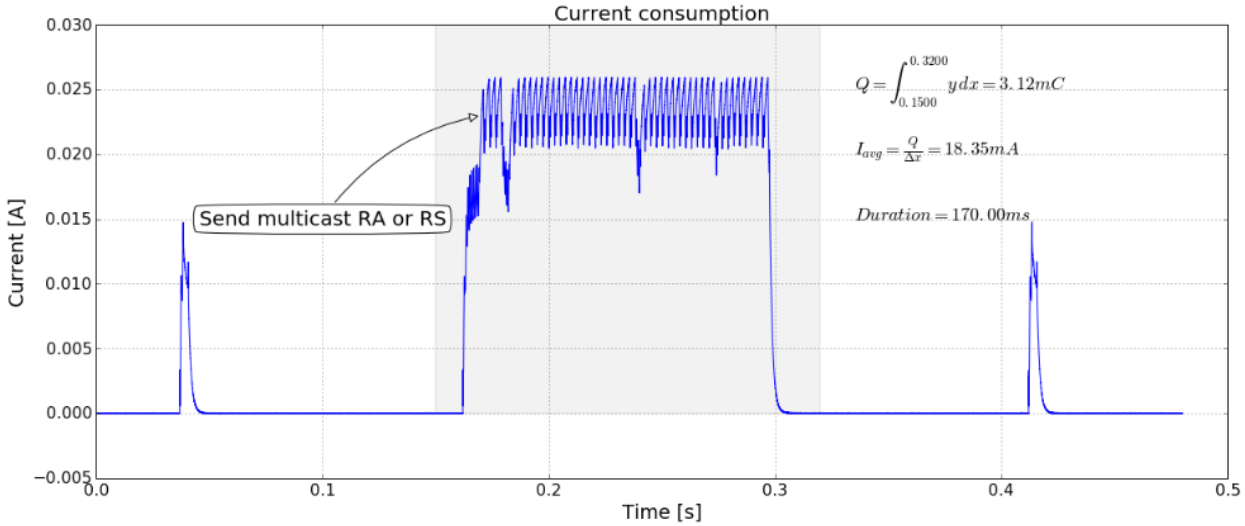

Figure 13. Current consumption during a multicast RA or RS message.
The message is repeated by ContikiMAC during the full period of the RDC which is here 125ms.

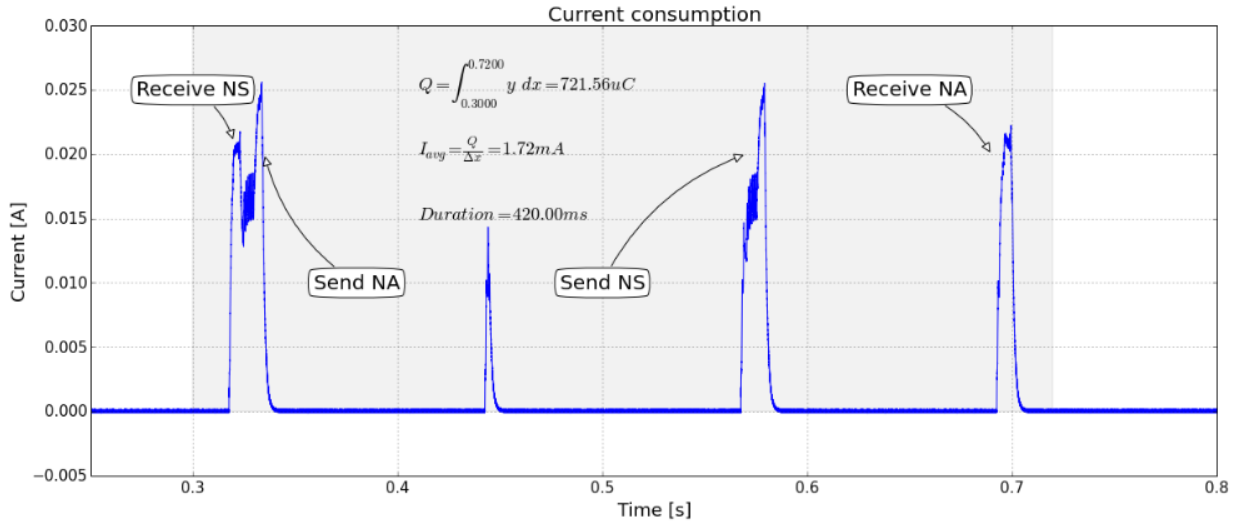

Figure 14. Current consumption during NS/NA message exchange for an interface.

Using the current consumption measured in Figure 13 and Figure 14, we compute the total charge consumed by the exchange of the periodic ND messages for different intervals over a period of 24 hours. The number of occurrences for the NS/NA messages is doubled because there is a minimum of 2 interfaces configured in the device for the local and global address.

Running on a battery with a capacity of 50mAh, the energy spent due to the NUD in the worst case is 0.23% per day of the total charge for the host and 0.48% per day for a router.

	Interval in minutes	Number of occurrences over 24h	Total charge	Waste of energy per day for a 50mAh battery (180 C)
NS/NA (2 interfaces)	15	192	138mC	0.077%
	10	288	207mC	0.115%
	5	576	415mC	0.23%
RA	30	48	150mC	0.083%
	10	144	450mC	0.25%
RS	60	24	9mC	0.005%

Table 5. Waste of energy for periodic messages exchange in the standard NDP assuming no retransmission.

Energy evaluation of the optimized NDP

With the optimized neighbor discovery protocol there is no more periodic unsolicited multicast RA message neither periodic NUD with NS/NA message exchange between the interfaces. The current consumption is consequently reduced (Figure 15). The router and the host behaves similarly because both registers them self to a parent router and periodically update the registration before the address expires. The unit of the ARO option is in minute, therefore the minimum registration lifetime is 1 minute in the worst case. There is no limit for the maximum value expects the size of the field which is 16-bit.

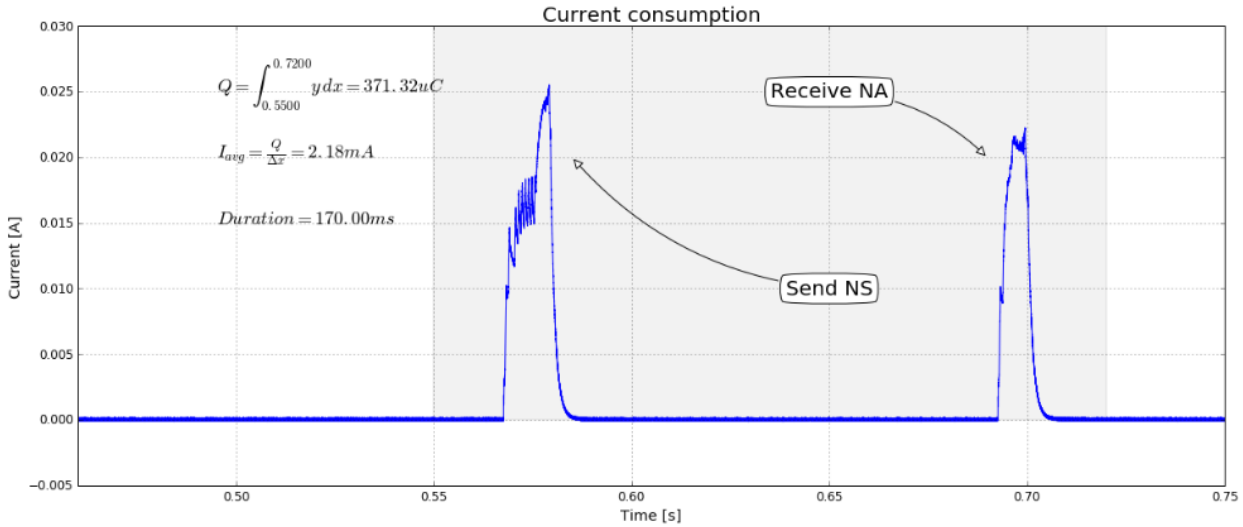

Figure 15. Current consumption during ARO registration.

$

	Registration lifetime in minute	Number of occurrences over 24h	Total charge	Waste of energy per day for a 50mAh battery
NS/NA	60	24	9mC	0.005%
	15	96	36mC	0.02%
	1	1440	534mC	0.3%
RS	60	24	9mC	0.005%

Table 6. Waste of energy for periodic messages exchange in the optimized NDP assuming no retransmission.

The comparison between the two protocols is not straightforward because the different lifetimes can vary from implementation to implementation. Using the value from Table 5 and Table 6, in the best case, the host spends 15x less energy per day with the optimized NDP considering only the NS/NA message exchange.

RPL with the optimized NDP

NDP is used to create the Neighbor Cache Entries (NCE) and to keep track of the reachability of the nearby nodes whereas the Routing Protocol for Low-Power and Lossy Networks (RPL) is used to build the mesh network and is used strictly for the routing purpose (Figure 16).

The RPL protocol implemented in Contiki-OS is slightly modified to work together with the optimized NDP. The current implementation of RPL allows creating NCE directly from the reception of certain RPL messages which make possible RPL to run without ND protocol. In this case a neighbor is assumed to be REACHABLE at any time. But it is required from the RPL specification that the reachability of a router be verified before the router can be used as a parent. When the standard NDP is activated with RPL, the neighbor remains in the REACHABLE state until its reachable timeout expires. When the timer expires, if the neighbor is a default router, instead of going to STALE it enters DELAY sate in order to force a NUD on it. Otherwise, if there is no upward traffic, the node never knows if the default router is still reachable. This mimics the optimized NDP behavior.

Since we've now implemented the optimized NDP, the reachability of the neighbors is handled by this latter and RPL doesn't need any more to mimic the behavior of the protocol for the reachability of the neighbors. We thus removed the possibility for RPL to add new neighbor from RPL messages by ignoring the message if the sender it's not already in the NCE. The RPL node use entries already available in the Neighbor Cache to create its parent.

The behavior of the router in the RPL instance is not altered and the node continues to send periodical DIO messages, respond to DIS, forward DAO and DAO ACK messages and take part in the routing. In contrast, a host is forced to behave as a leaf. As a leaf the host joins the network with an infinite rank, ignores DIS and DAO messages, sends DIO only when directly addressed with DIS and accepts DIO only from the routers it has registered with.

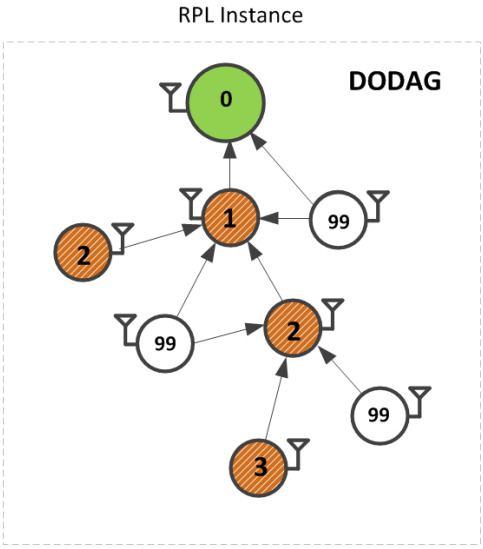

Figure 16. RPL Instance

Device Management and Information Reporting

To manage the sensors, on top of CoAP [7], we use the Open Mobile Alliance (OMA) Lightweight Machine To Machine (LwM2M) protocol [8]. CoAP provides a request/response interaction between application endpoints and includes key concepts of the Web such as URIs and Internet media types but doesn't ensure interoperability on the application layer. OMA LwM2M was designed for this purpose and respond to the demand of a common standard for managing low power devices in constrained networks.

Interfaces

In OMA LwM2M, the sensor is an LwM2M Client and connects to an LwM2M Server as opposed to CoAP where the sensor is the server from which a CoAP client reads the resources. LwM2M provides four main functionalities designated as interfaces, each designed for specific purpose:

- **Bootstrap** - used to provide LWM2M management server credentials to a LWM2M client for registration purposes.
- **Client Registration** - enables the LWM2M client to register with one or more LWM2M servers.
- **Device Management and Service Enablement** - used by the LWM2M Server to access an

Object Instance or Resource on the client side.

- **Information Reporting** - used by the LWM2M Server to observe any changes in a resource on a LWM2M client and to receive notifications from the client when an observed value changes.

Interfaces operations

Each of the interfaces listed above provides their own operations and are mapped to specific CoAP method with its own Content Format and Response Code. This extends the four default methods of CoAP which are GET, PUT, POST and DELETE to 17 operations (Table 7) made available by the LwM2M v1.0 standard for the interaction with the devices. We're using 9 of them (in bold in Table 7) to interact with our device and do not uses at all the Bootstrap interface.

Interfaces	Operations
Bootstrap	Bootstrap request, Write, Delete, Bootstrap Finish
Client Registration	**Register**, **Update**, De-register
Device Management & Service Enablement	**Read**, **Write**, Write Attributes, **Execute**, Create, Delete, **Discover**
Information Reporting	**Observe**, **Cancel Observation**, **Notify**

Table 7. Operations provides by each interface.

The OIR Model

With the OMA LwM2M Object Model, a resource is exposed by the LwM2M Client for consumption by the application through an URI in the following format: *Object ID / Instance ID / Resource ID.* The structure consists of three unsigned 16-bit integers separated by the character '/'. Objects are typed containers, which define the semantic type of instances. Instances represent specific object types at runtime, and allow endpoints to expose multiple sensors and actuators of a particular type. Object instances are themselves containers for resources, which are the observable properties of an object.

An object is defined with the properties listed in Table 8.

Properties	Values	Description
ID	0 to 32768	Object identifier
Name	*String*	Object name
Instances	Single or Multiple	Indicate if multiple instances are allowed
Mandatory	Mandatory or Optional	Indicate if object is mandatory

Object URN	urn:oma:lwm2m:oma:{*Object ID*}	Specify the object URN

Table 8. Object properties.

The ID of an object is categorized into the following classes by the standard:

- 0 – 1023 : Object produced by OMA
- 1024 – 2047 : Reserved
- 2048 – 10240 : Object produced by Standards Development Organizations
- 10241 – 26240 : Object produced by vendors or individuals (range accepts registration)
- 26241 – 32768 : Object produced by vendors or individuals (no registration required)

A resource, which is the final object of interest because it contains the value, is defined with the following properties:

Properties	**Values**	**Description**
ID	0 to 32768	Resource identifier
Name	*String*	Resource name
Operations	Read (R), Write (W), Read/Write (RW), Execute (E)	Which accesses the resource supports
Instances	Single or Multiple	Indicate if multiple instances are allowed
Mandatory	Mandatory or Optional	Indicate if resource is mandatory
Type	Integer, Float, String, Time, Boolean, Opaque, Objlnk	The type of the resource
Range or Enumeration	*String, Numbers*	Limits de value of the resource
Units	*String*	Specifies the units of the resource value

Table 9. Resource properties.

The ID of the resource is categorized into the following classes by the standard:

- 0 – 2047 : Common resources which are unique only inside object
- 2048 – 32768 : Reusable resources which are unique for all objects

For instance, the Resource ID 1 (<2048) in the Object ID 1 is not the same resource as in the Object ID 3 whereas the Resource ID 5700 (>2048) has the same meaning inside all objects.

Objects definition

The OMA LwM2M v1.0 defines its own Objects for the basic interaction with any LwM2M Client.

Object	**Object ID**	**Instance**	**Mandatory**
LWM2M Security	0	Multiple	Mandatory

LWM2M Server	1	Multiple	Mandatory
Access Control	2	Multiple	Optional
Device	3	Single	Mandatory
Connectivity Monitoring	4	Single	Optional
Firmware	5	Single	Optional
Location	6	Single	Optional
Connectivity Statistics	7	Single	Optional

Table 10. Objects produced by OMA.

The IPSO Alliance has defined other Object IDs describing sensor devices by publishing the IPSO Smart Objects Start Pack which introduce 18 Smart Objects like temperature sensor, presence sensor, etc. To complement this initial set of objects, the IPSO Smart Object Expansion Pack adds 16 common templates sensors, 6 special template sensors, 5 actuators and 6 control switch types.

Implementation

We're using the available implementation of the OMA LwM2M standard within Contiki 3.x by adding our own Objects and Resources when necessary. We complete the registration process with the server and implement the Registration Update mechanism which is the key feature for sleepy nodes since we deactivate the LPL.

LwM2M Server

The open source applications for the server side are still not widely available because the standard is relatively recently published. Leshan is a popular LwM2M server implemented in Java which is based on the Californium project for the CoAP implementation. Wakaama and AwaLwM2M are both implemented in C and provide API to build a server without the need of an intimate knowledge of the M2M protocol. Wakaama is based on the Erbium CoAP library as Contiki but the library was modified to run on Linux. AwaLwM2M is using the libcoap library.

To manage a bunch of devices one need a GUI and Leshan is the only one that actually provides a Web interface to interact with LwM2M Clients. However we needed some customization on the interface and we find that there is an easier solution to develop a Web interface than Java.

We've developed our web interface with NodeJS using the *lwm2m-node-lib* module available at https://github.com/telefonicaid/lwm2m-node-lib. The module implements the Client Registration, the Device Management & Service Enablement and the Information Reporting interfaces where each of the operations of the protocol are made available through JavaScript call which make the development of the web interface very straightforward.

We've slightly modified the core functionality of the module and enabled the possibility to queue the operations in destination to a device that was registered with the "Q" binding. The operations will be delivered to the device when this later wakes-up from the sleep state and update its registration to the server.

The Web Interface

Our LwM2M Web Server provides actually 4 main views:

- **Devices list**: centralized view of each registered LwM2M Clients with their IP address, creation date, binding and a dynamic lifetime count down (Figure 17).
- **Device details**: detailed view of an LwM2M Client with its Objects, Instances and Resources. The view provides buttons to interact with each Resource within an Object and a list of current Observations on the device (Figure 18).
- **Objects DB**: manage Objects and Resources ID (Figure 19).
- **Device models**: manage device models (Figure 20).

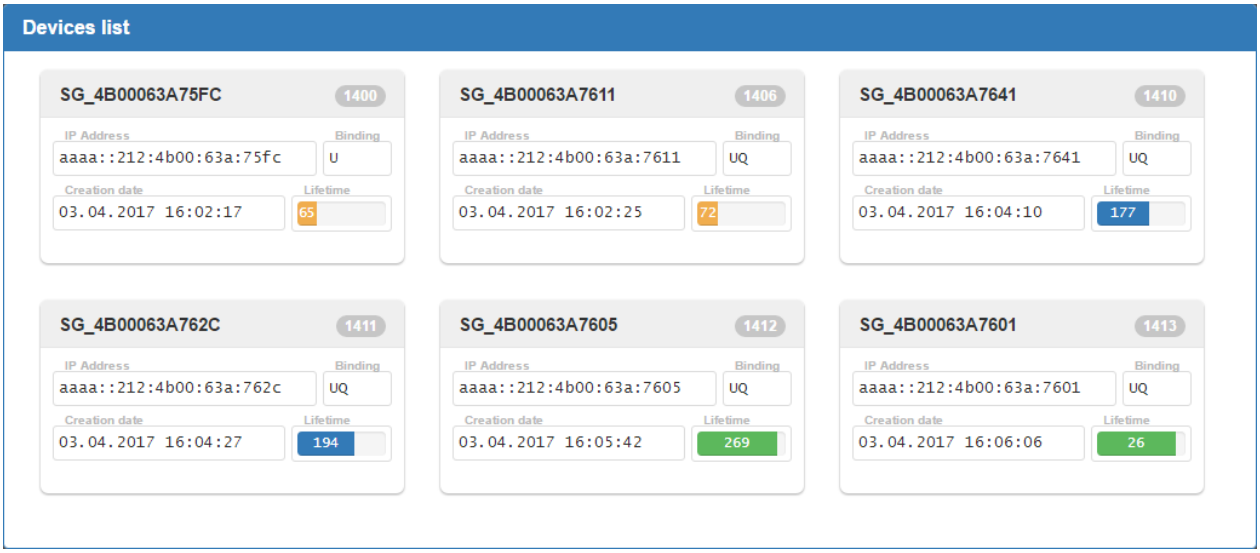

Figure 17. View of the registered LwM2M Clients on the LwM2M Server. The device's card shows the remaining lifetime of the device before it should update its registration.

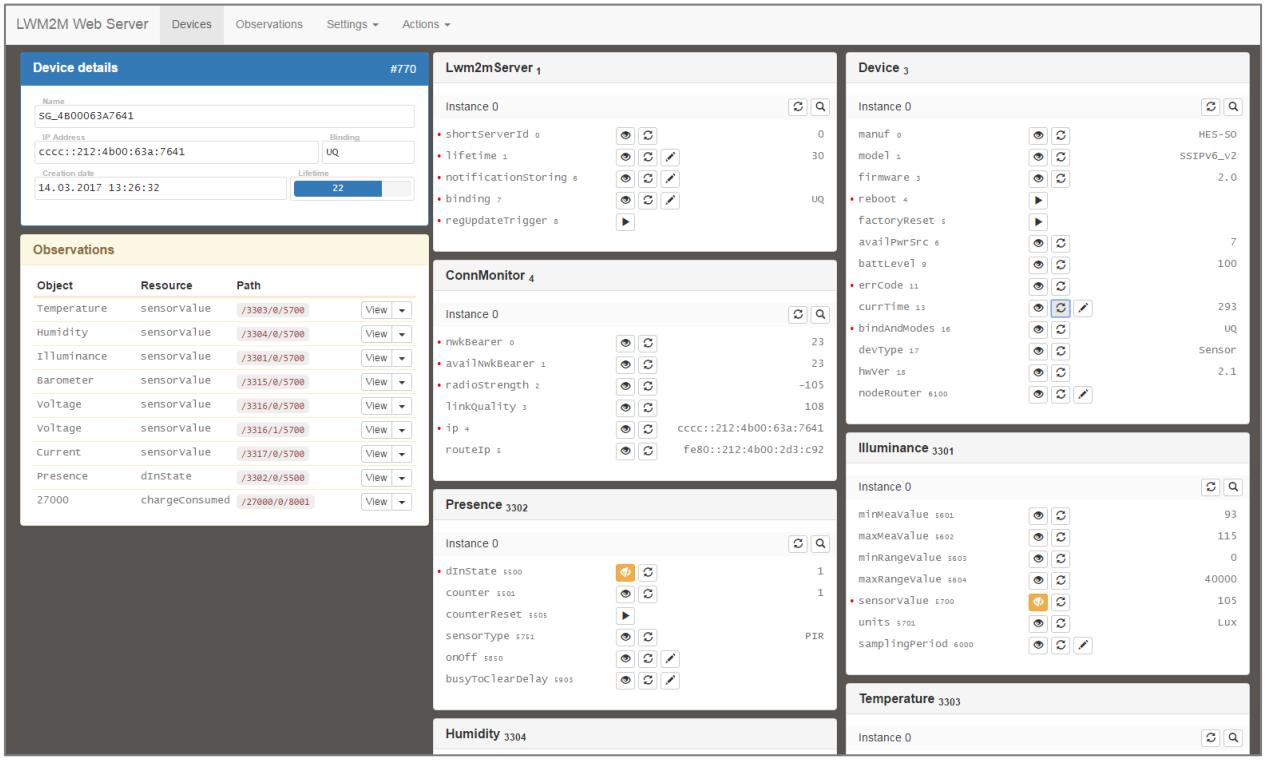

Figure 18. View of the LwM2M Web Interface for the management of an LwM2M Client.

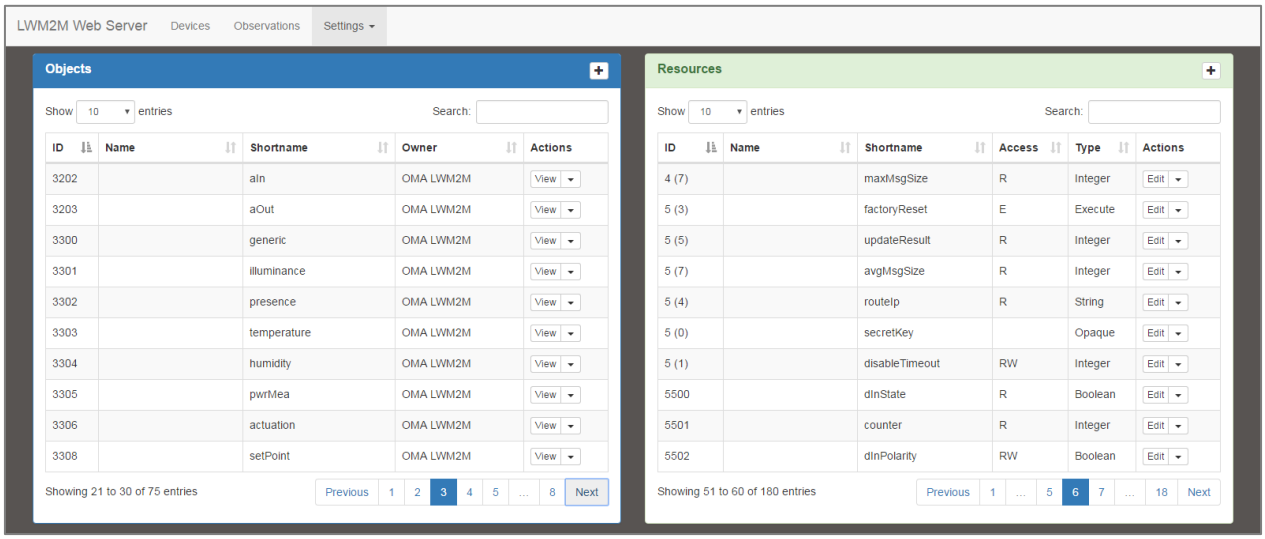

Figure 19. Objects and Resources Database.

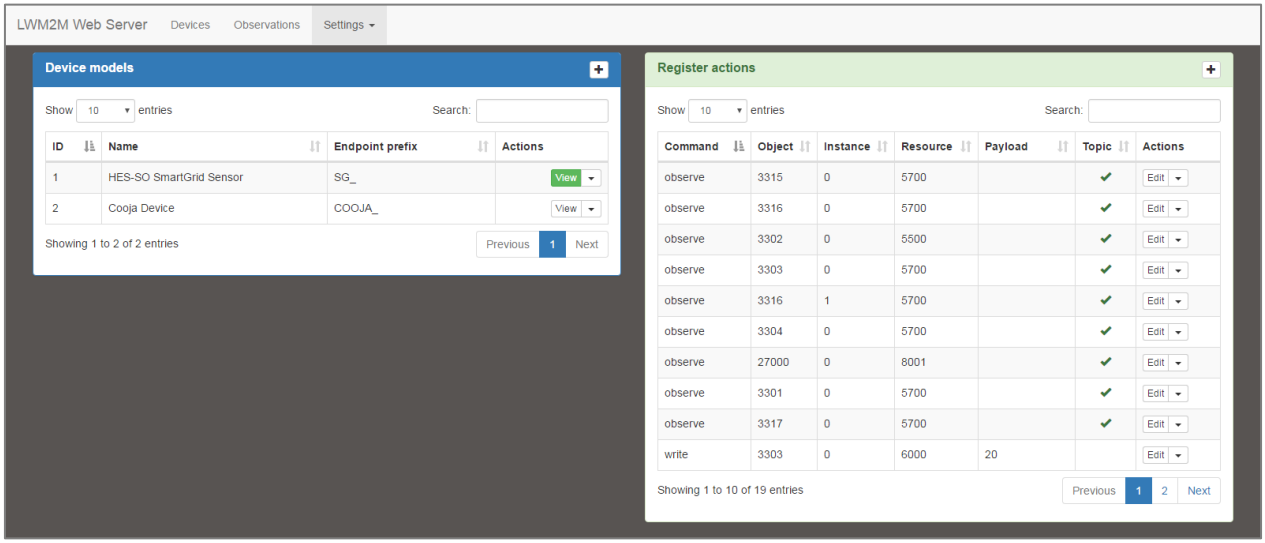

Figure 20. Device model

Client registration

Once the device has found a router and joined an RPL instance, it tries to register to the LwM2M Server by sending a Register request. The server responds with the new registered device ID in the message. To maintain the registration, the ID is used by the client to update its registration on the server before its registration lifetime expires. The Figure 21 shows the Registration and Update messages exchange. When registering to the server, the client sends its name, its lifetime, its binging and the list of Objects and Instances available on the device. The “U” in the binding means that the client is reachable via the UDP binding. The “Q” means that the server must queue all requests to the client and send them when this latter goes online. We use the Queue mode for the leaf devices in the network since there are sleeping most of the time.

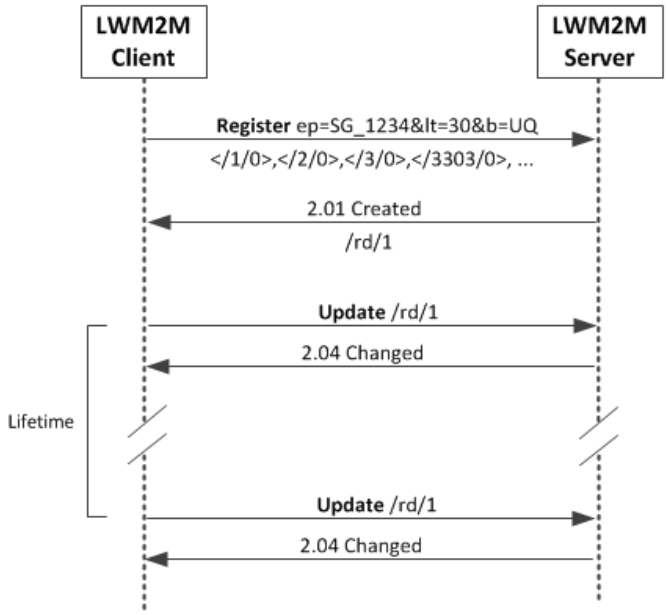

Figure 21. Client Registration & Update message exchange.

The Web Application is using the payload in the Register message to create the device with its associated Objects and Instances and create a view allowing the user to interact with this latter (Figure 22). The Objects and Resources ID are automatically mapped to their corresponding name by looking in the Objects and Resources database of the web application. The user can add its own Objects and Resource in this database so that the Objects and Resources from its proprietary device are automatically recognized (Figure 23). When an Object is recognized its mandatory Resources are automatically added in the view. The web interface shows the mandatory resources with a red dot in front of the name.

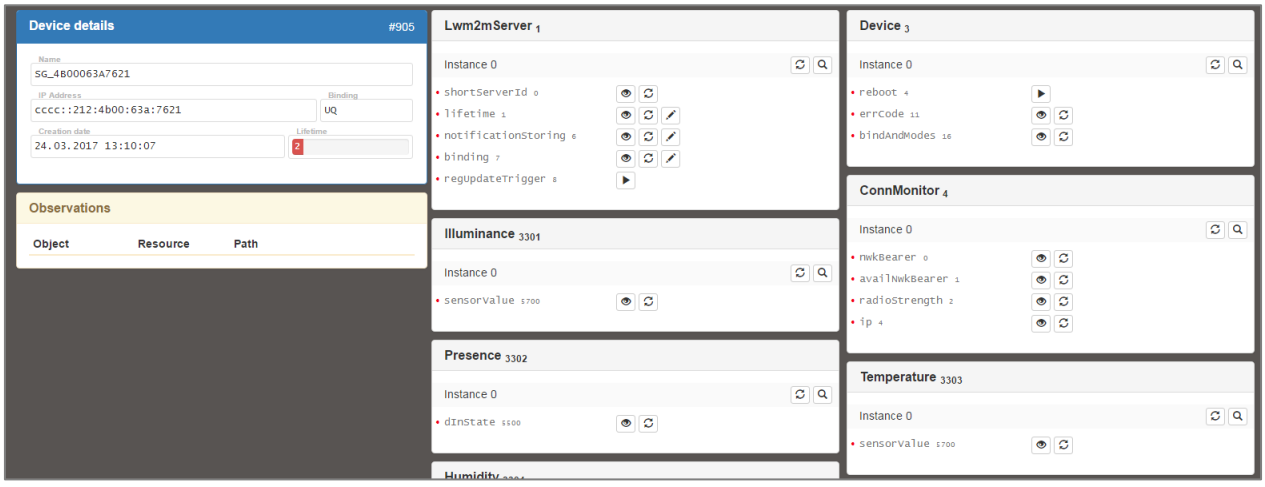

Figure 22. When a client registers to the server, the web application provides a view with the available Objects in the device and creates automatically the mandatory resources for the Object even if they are not available on the device.

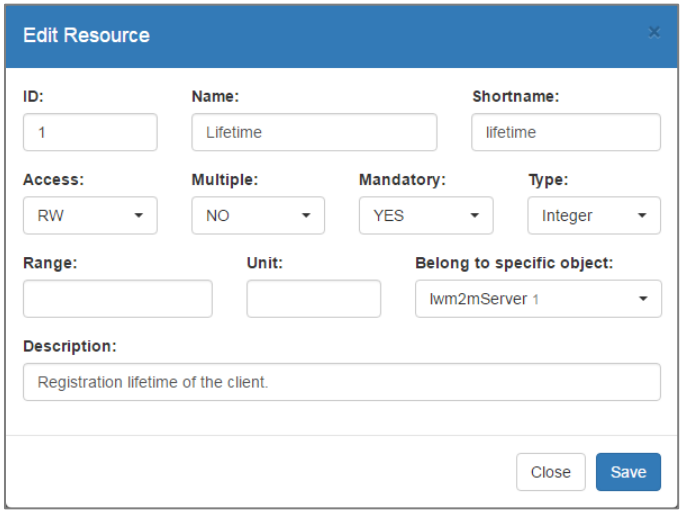

Figure 23. Add and Edit Resource window.

Device management & Service Enablement

To interact with an Object the web interface provides a panel for each Object available in the client (Figure 24). To find more about an Object, one can use the "Discover Instance" to retrieve from the client the Resources which are not mandatory. The "Read Instance" button send a read operation for the whole instance and retrieve the value for each Resource of the Object.

Note than the LwM2M implementation in Contiki currently does not allow to retrieve with the Read Instance operation the value of a resources where the data need to be processed and the value returned from a function.

Each row in the Object panel corresponds to a Resource. The buttons available to interact with a Resource depends of the type and the configured properties of this Resource in the "Objects and Resources" database.

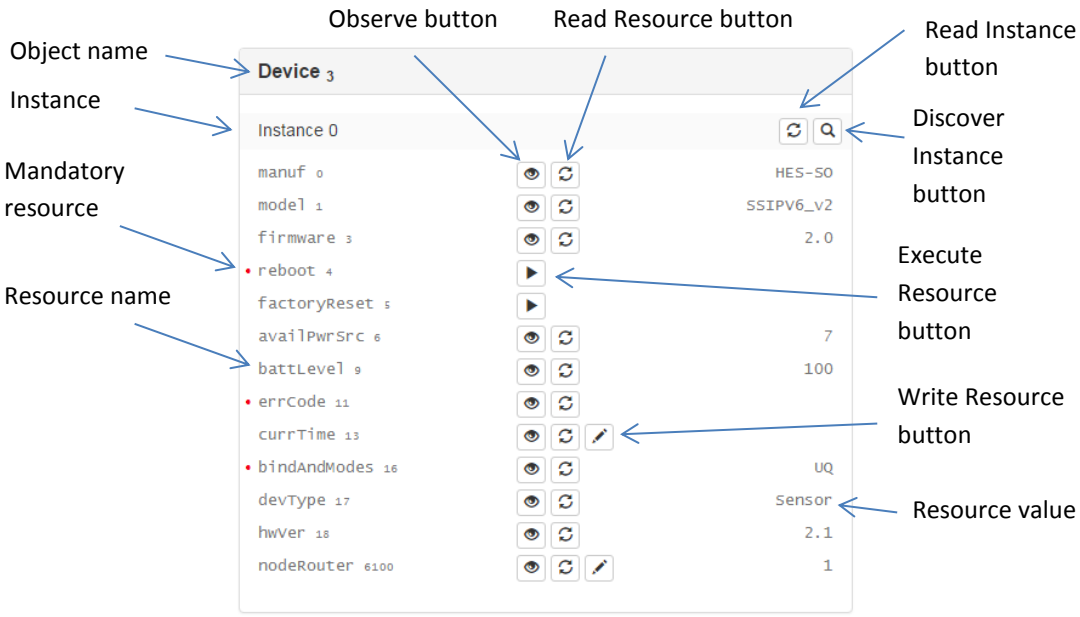

Figure 24. Object panel where each row represents a Resource.

Every readable Resource is shown with the Observe button  but not all of these resources are observable. In fact, the LwM2M standard v1.0 doesn't describe actually any mechanism during the discovery operation which tells the server that a Resource is observable or not. But this features can be easily implemented and not impede any malfunctioning of the protocol. Since the protocol is using the Core Link Format, when the Client responds to a Discover operation, the client can embed the "obs=1" string for a particular resource to inform the server that a particular resource is observable. The content of the discover responses would be for example as follow: `<3303/0/5601>,<3303/0/5602>,<3303/0/5700>;obs=1,<3303/0/5701>`.

Automatic device configuration

Once registered to the LwM2M Server, our Web Application recognizes the client as a registered Device Model based on the endpoint name sent during registration process and starts to configure automatically the client with the registered operations configured for the Device Model. The user can register any of the LwM2M operations via the Device Models interface (Figure 25). These automatic operations consists mainly to configure the sampling period for every sensors and to enable the observation of the sensor's value resources. We furthermore added the possibility to configure an MQTT Topic for the "observe" operation which will be used to forward the sensor's values to an external database. The Figure 27 shows the datalog of the sensors value via MQTT to an external database. The device has stopped working at mid-night due to the battery depletion but at 10h o'clock the device had sufficient solar energy to power on, join the network and restart the data logging of the configured resources thanks to the auto-configuration from the Web Application.

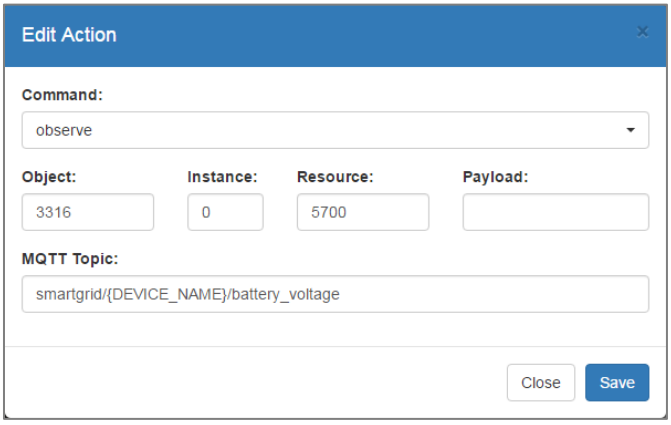

Figure 25. The user can configure an automatic LwM2M operation to be sent to the client after the Registration process.

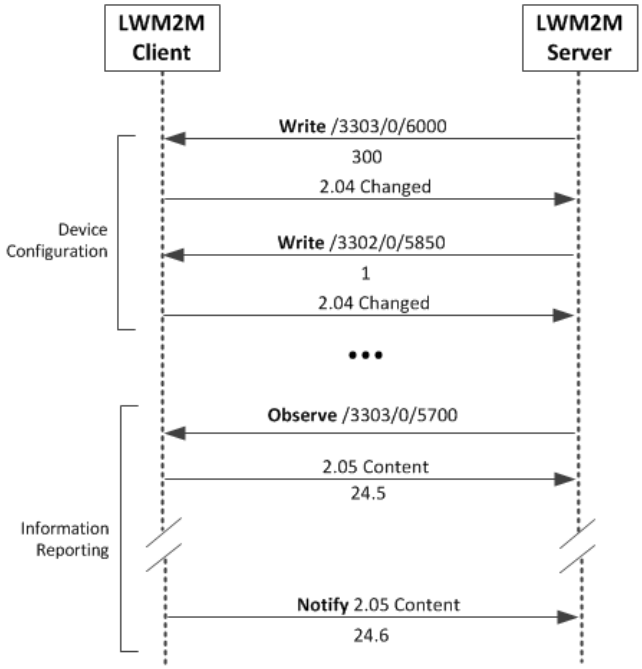

Figure 26. The Web Application configures automatically a Client by sending after registration the operations from Device Management & Service Enablement and Information Reporting interface.

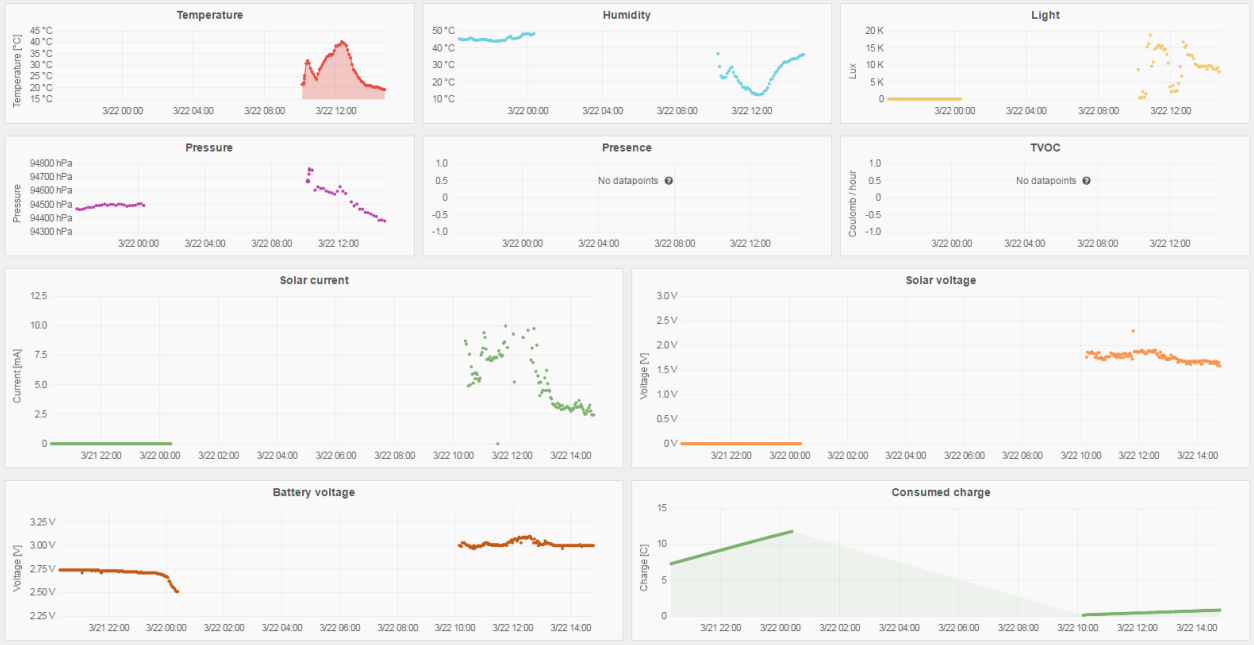

Figure 27. Datalog of the sensors value via MQTT to an external database. The plot of the battery voltage shows that the device has stopped the data logging at around mid-night due to the battery depletion. At 10h the device has joined the network again and restarted the data logging thanks to the autoconfiguration from the Web Application.

Desynchronization of the sampling period

We've implemented an algorithm which desynchronizes the sampling interval of each observable resource. In fact since the observation of the sensor's resources are started at the same time immediately after the registration process, it is likely that the measure of the sensor's value will be executed at the same time leading to consequent voltage drop if many sensors are enabled and read consecutively. To let the battery relax, our algorithm calculates the best interval between the sampling and adjusts the execution time of the measure based on this interval. Each time the observation of resource is started or the sampling period of the sensor is updated, the best interval is recalculated and timers adjusted. The following formula is used to calculate the best interval:

$$BI = \frac{gcd([I_1, I_2, \dots, I_n])}{n} \; seconds$$

where $\gcd()$ is the Greatest Common Divisor function, n the number of intervals and $[I_1, I_2, \dots, I_n]$ the list of each interval.

Power consumption analysis

The current consumption of each periodic event (LwM2M Information Reporting) has been measured and reported in Figure 28 to Figure 36. The total charge Q and the average current during the task are calculated by integrating the curve's data over the duration of the task. The

shape of the current for each task looks very similar and consists mainly to wake-up the processor, process the event, power on the sensor, start the measure during which the processor goes in deep sleep mode and finally send the data over the radio. With these measures we're able to calculate the total consumed charge while performing sensor measure at any interval.

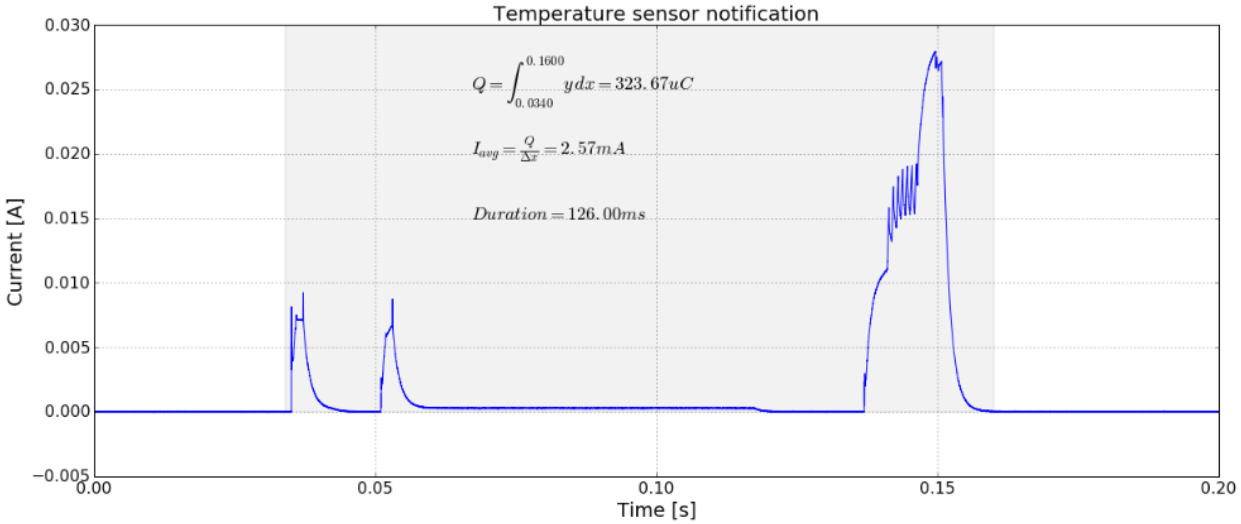

Figure 28. Current consumption of the IPSO Temperature measure and notification.

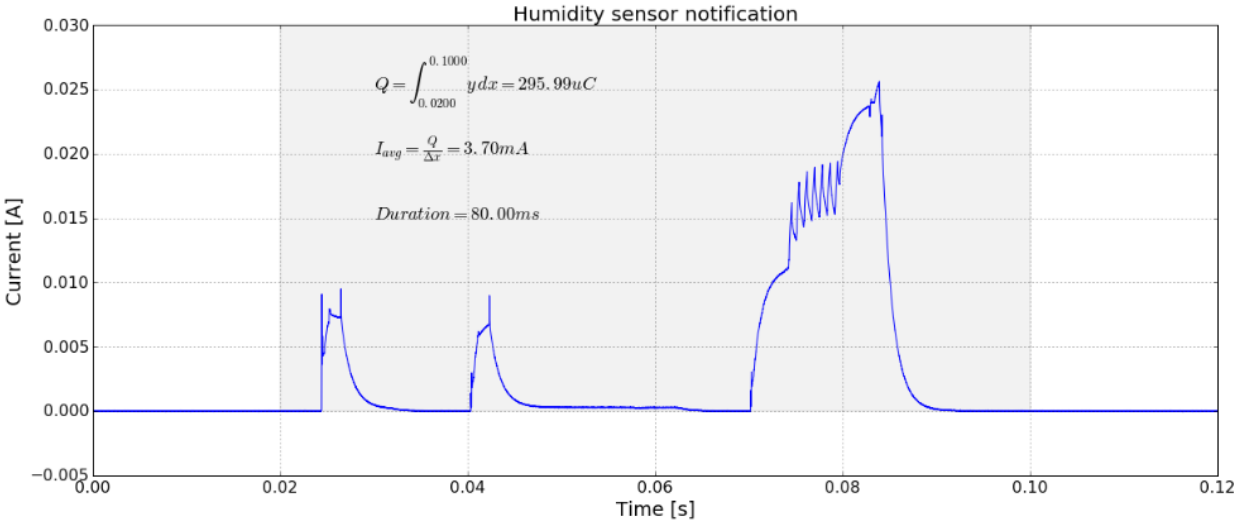

Figure 29. Current consumption of the IPSO Humidity measure and notification.

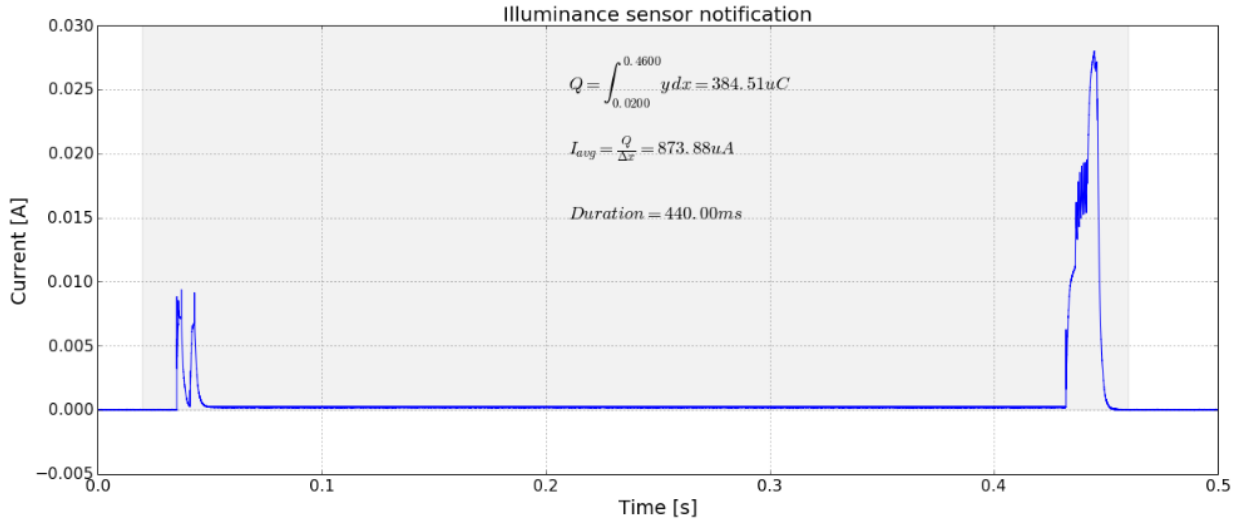

Figure 30. Current consumption of the IPSO Illuminance measure and notification.

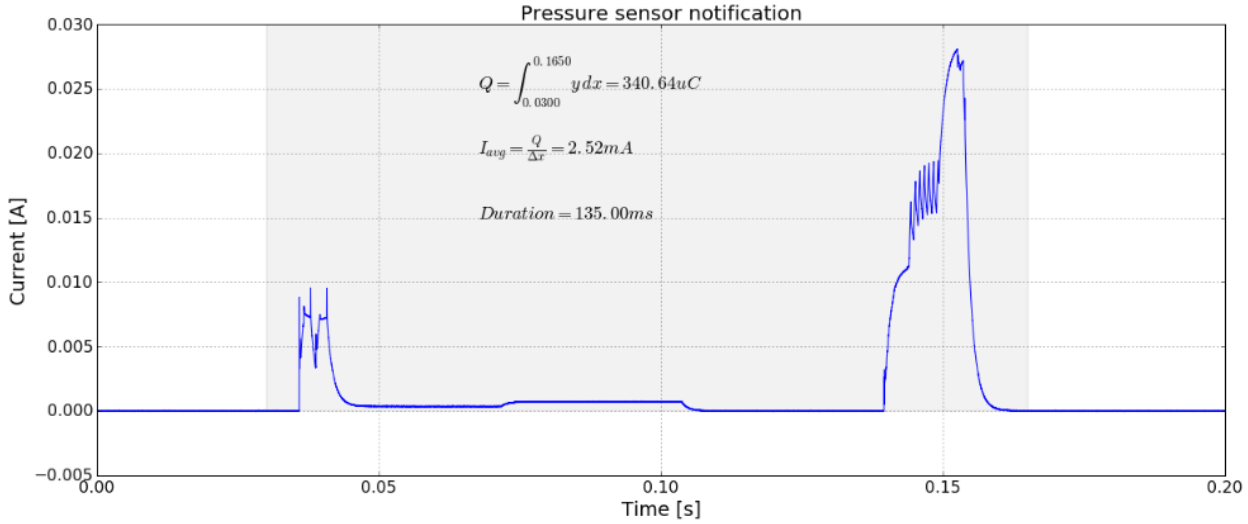

Figure 31. Current consumption of the IPSO Barometer measure and notification.

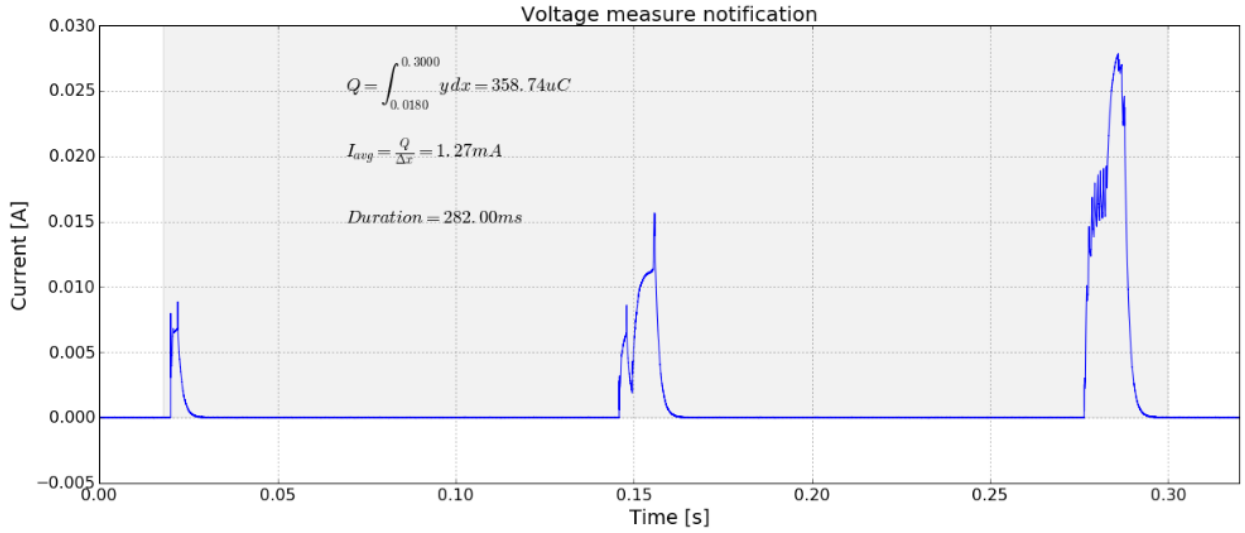

Figure 32. Current consumption of the IPSO Voltage measure and notification.

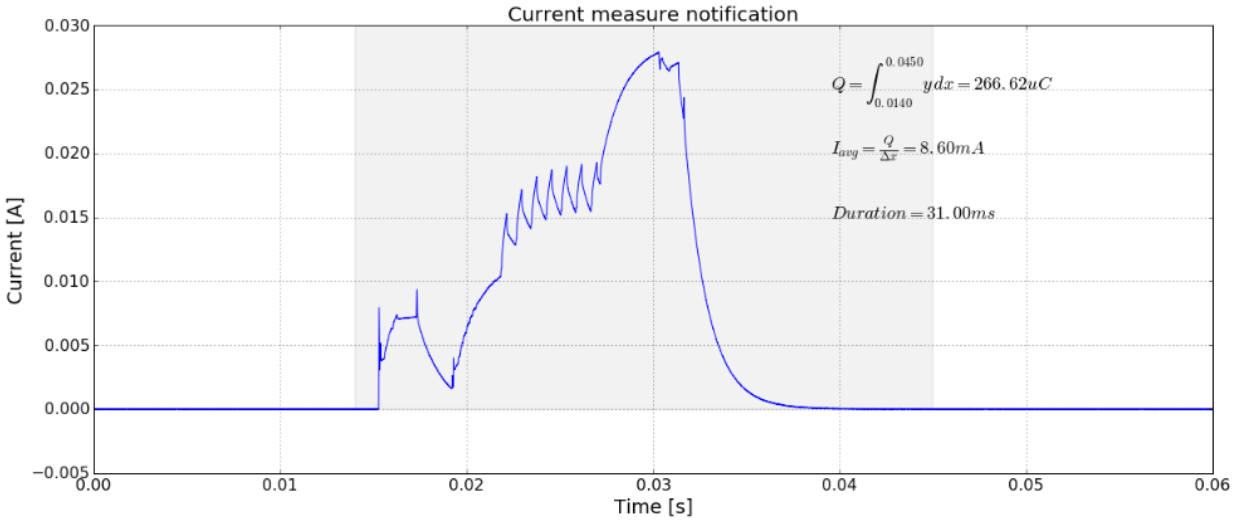

Figure 33. Current consumption of the IPSO Current measure and notification.

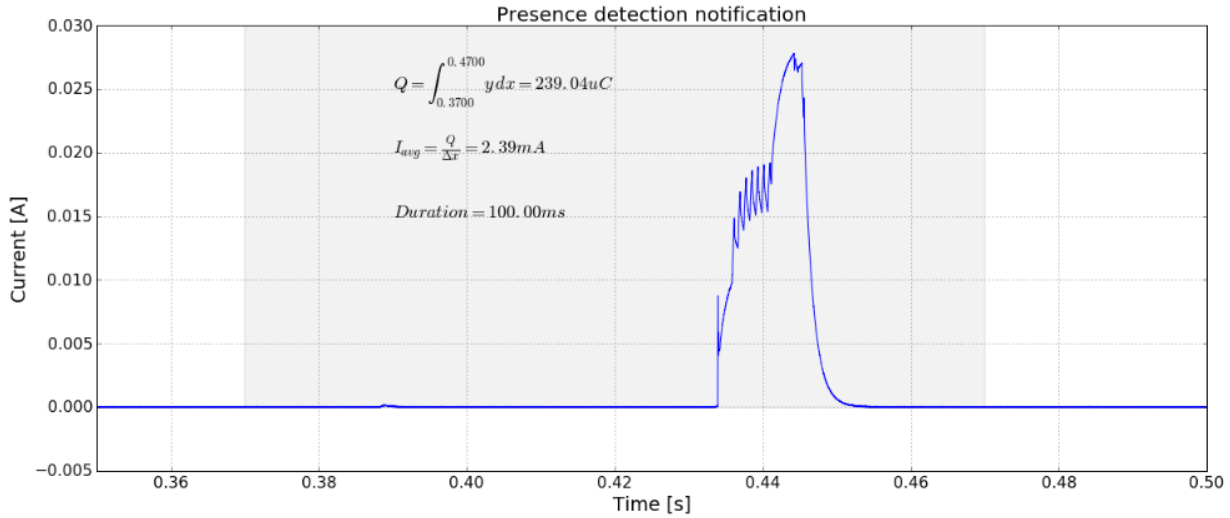

Figure 34. Current consumption of the IPSO Presence detection and notification.

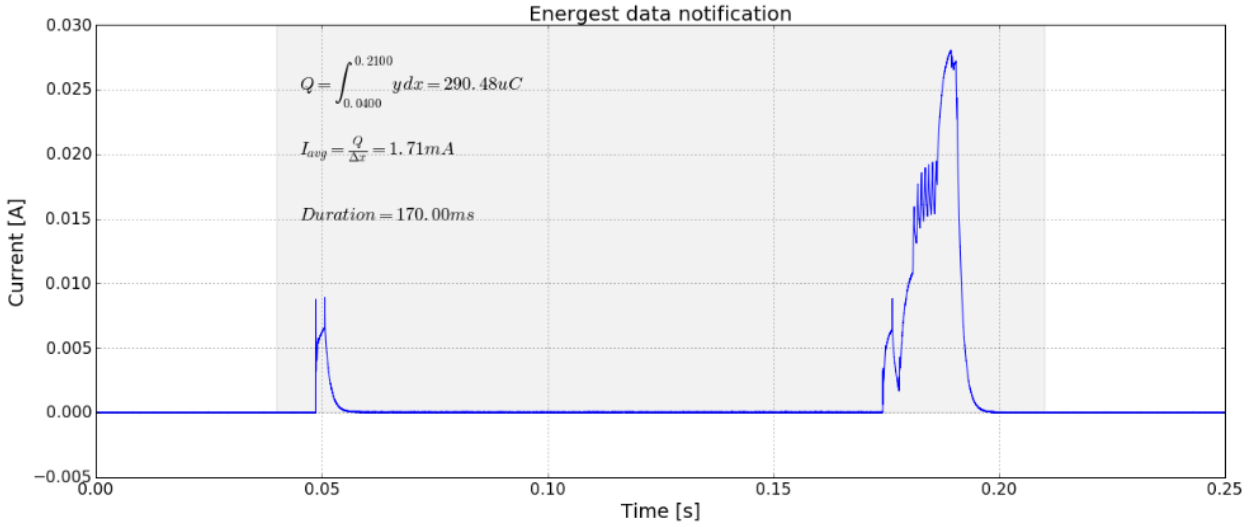

Figure 35. Current consumption of the HES Energest computation and notification.

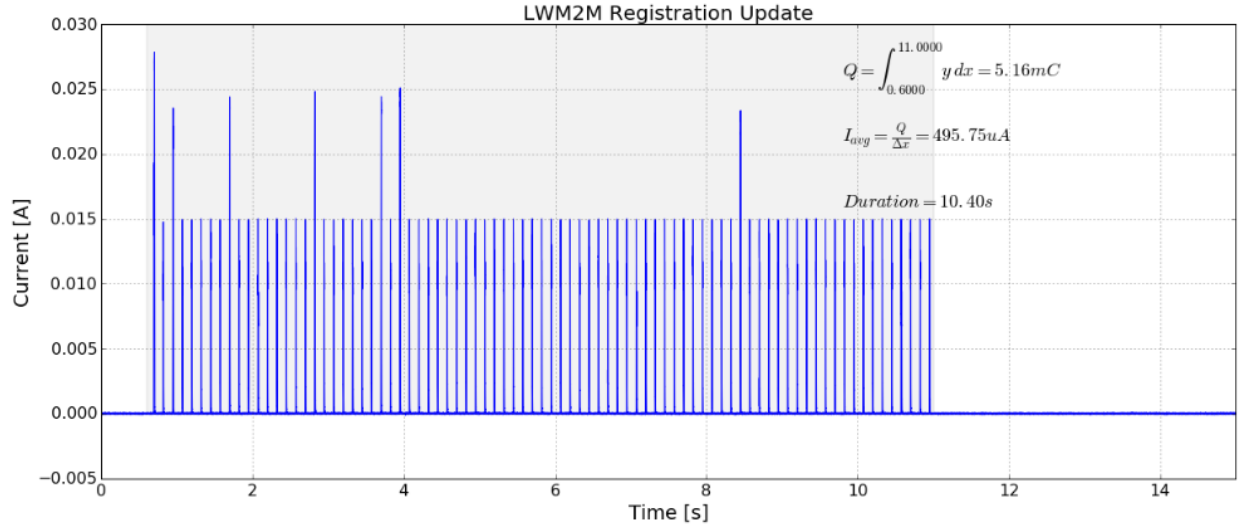

Figure 36. Current consumption of the OMA LwM2M Registration Update with the RDC activated for 10 seconds to receive the queued messages on the server.

The total charge of each periodic event is reported in Table 11. The charge C is almost identical for every sensor measurement and approximate 300µA in average while the LwM2M Registration Update consumes the most because the RDC is activated for 10 seconds. The duration of the RDC activation was choose arbitrary and can be reduced for more optimization. We choose 10 seconds to give sufficient time for the farthest device from the sink to receive every pending message on the server.

To determine the lifetime of the device we compute the average current consumption of the device while performing sensors measurement at different intervals. The same period of measurement was set for each sensor and assuming a presence detection at the same interval. The average current consumption I_{avg} is reported in Figure 37. With an interval of 5 minutes, the

avg. current consumption is 32.2µA while performing a Registration Update every 5 minutes (Figure 37). The device can last on a battery of 50mAh (using 80% of the capacity) theoretically about **54 days** (1300 hours) (Figure 38).

Periodic event	**Charge**	**Duration**	**I_{avg}**
Temperature measure	324 µC	126 ms	2.57 mA
Humidity measure	296 µC	80 ms	3.70 mA
Illuminance measure	384 µC	440 ms	0.87 mA
Pressure measure	340 µC	135 ms	2.52 mA
Battery voltage measure	359 µC	282 ms	1.27 mA
Solar voltage measure	359 µC	282 ms	1.27 mA
Solar current measure	266 µC	31 ms	8.6 mA
Presence detection	240 µC	100 ms	2.4 mA
Energest reporting	290 µC	170 ms	1.71 mA
ARO Registration	371 µC	170 ms	2.18 mA
LwM2M Registration Update	5 mC	10 s	0.5 mA

Table 11. Total charge C consumed by each periodic event.

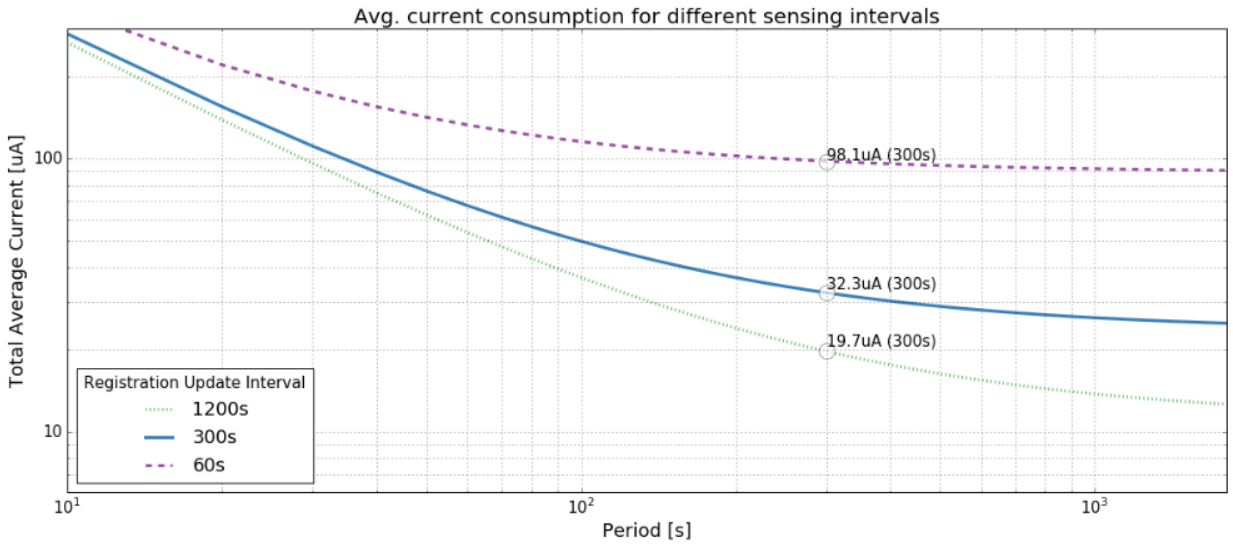

Figure 37. Average current consumption of the device while performing sensor measurement at different period (same for all sensors) with a fixed ARO Registration period of 1 hour and LwM2M Reg. Update at 60, 300 and 1200 seconds. The deep sleep current of the device was set to 7µA.

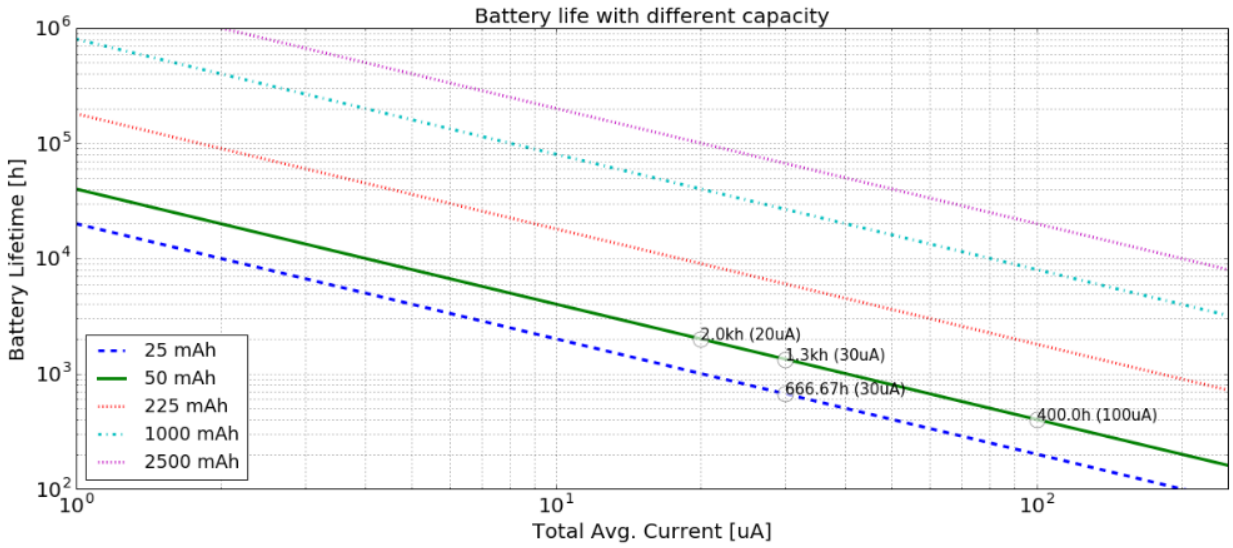

Figure 38. Battery lifetime for different average current consumption using 80% of the total battery capacity.

Battery recharge

To recharge the battery, the incoming power from the solar panel must be greater than the average power of the system load. We need to take in account the efficiency of the energy harvesting chip in the equation:

$$P_{solar} \cdot \eta_{PMU} > P_{system}$$

With P_{solar} the power coming from the solar panel, η_{PMU} the efficiency of the power management unit and P_{system} the power consumption of the system. The powers the solar panel needs to exceed to recharge the battery for different power consumption of the system are resumed in Table 12.

Condition	Power consumption of the system	Minimum solar cell power to exceed	
		In good light condition ($\eta_{PMU} = 0.8$)	In low light condition ($\eta_{PMU} = 0.6$)
1. Motion detector disabled	15µW	> 19µW	> 25µW
2. Motion detector enabled	22µW	> 28µW	> 37µW
3. Motion detector enabled, Information Reporting (interval of 5min except for power measurement at 2 min) (38 µA avg. current)	~106µW	> 132µW	> 177µW

Table 12. Power required from the solar panel to recharge the battery based on the equation above

Thanks to the illuminance sensor and the power measurement chip we're able to track the incoming power from the solar panel and the recharge of the battery. The plots in Figure 39 show that the main boost converter of the charging unit is turned on at very low light condition (~100 lux at 8h00). The system voltage in Figure 39 is the output voltage of the charging unit

which is either the battery voltage when there is no other input power (before 8h in Figure 39) or the boosted input voltage from the solar panel. When the system voltage starts to increase that means that the boosted input voltage is greater than the battery voltage and thus the recharge of the battery occurs.

The Figure 40 shows the recharge of the battery in very good light condition (> 2000 lux) where the battery is charged at about 2mA in average during 4 hours which is 16% of the total battery capacity. This record was realized with the device placed very close to the window at a very sunny location. In order to sustain the daily consumption, the device needs to be placed at a position where there is sufficient light during the day. The amount of daily input current required from the solar panel depends on the interval set for the sensors measurement and thus on the average current consumption of the system. The Figure 41 shows the solar input current required during certain duration to sustain the daily consumption of the device. For example, for the third condition from Table 12 where I_{avg} is 38uA, the input solar current needs to be at least 240uA during 4 hours to sustain the consumption of the day. The tracking of the solar panel input power allows in the long term to verify that the device is well positioned or not.

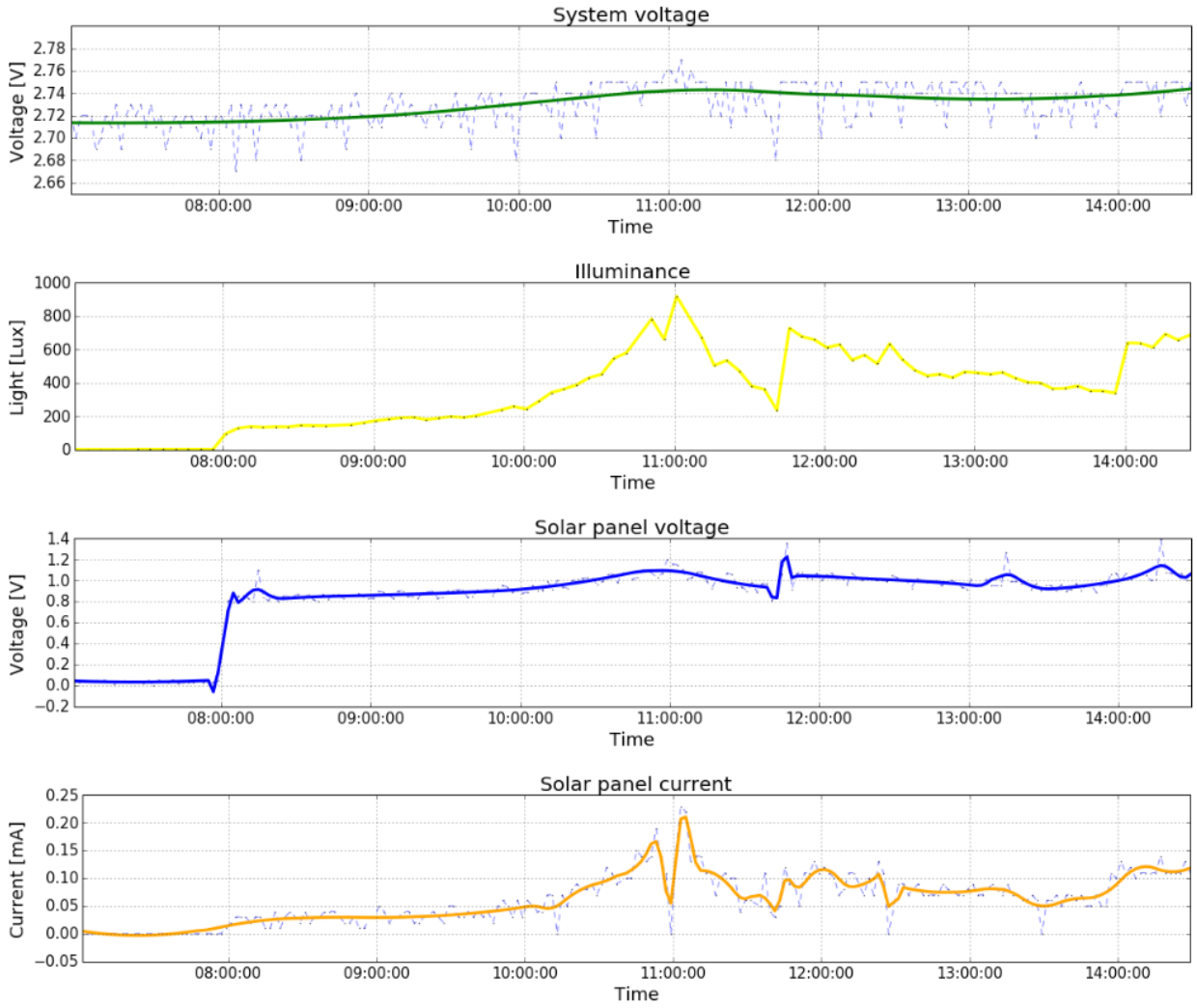

Figure 39. Recharge of the battery at very low light condition.

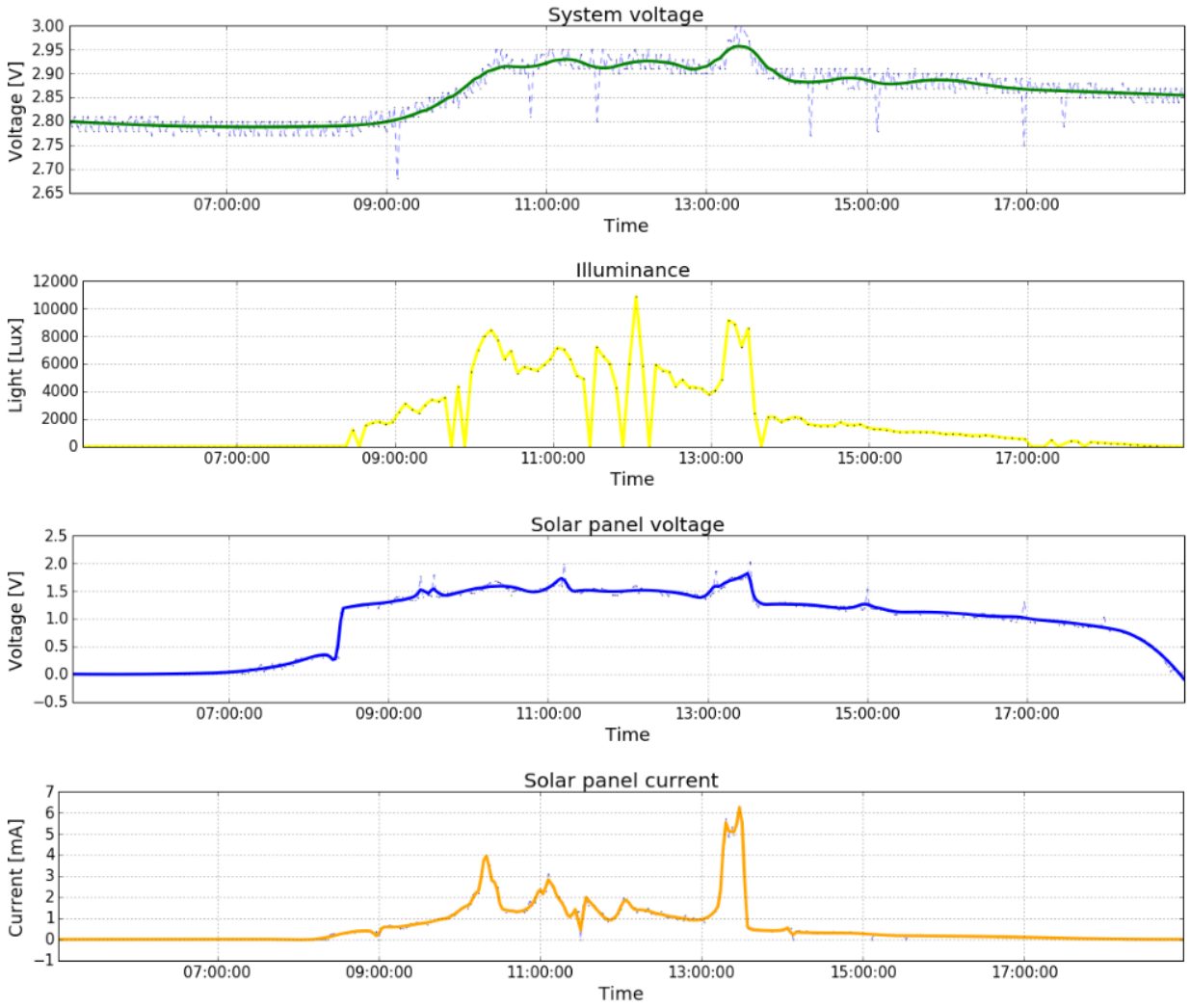

Figure 40. Battery recharge at good light condition.
The battery is recharged with 2mA in average during 4 hours which is 16% of the total battery capacity.

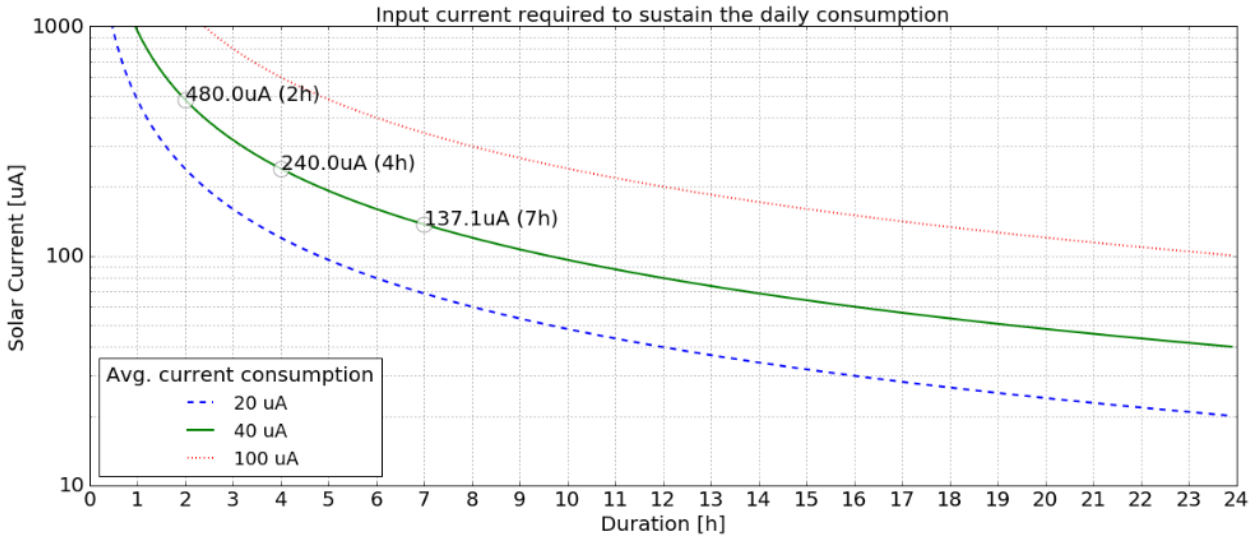

Figure 41. Input current with duration required from the solar panel to sustain the daily consumption of the device for different average current consumption.

Power consumption tracking system

Contiki-OS provides an energy estimation module which enables the tracking of the time spent by the software in the different tasks (CPU on, Power down mode, TX/RX, sensors, etc.). To estimate the energy consumption of the device we multiply the duration of a particular task by the average current draw during this task. The total energy consumption E is then defined as [9]:

$$\frac{E}{V} = \sum_i I_i \cdot t_i$$

where V is the supply voltage, I_i and t_i being the current and time used by a particular task. The average current used by each tasks are measured precisely as shown in Figure 42 and reported in the software for the calculation of the total charge consumption.

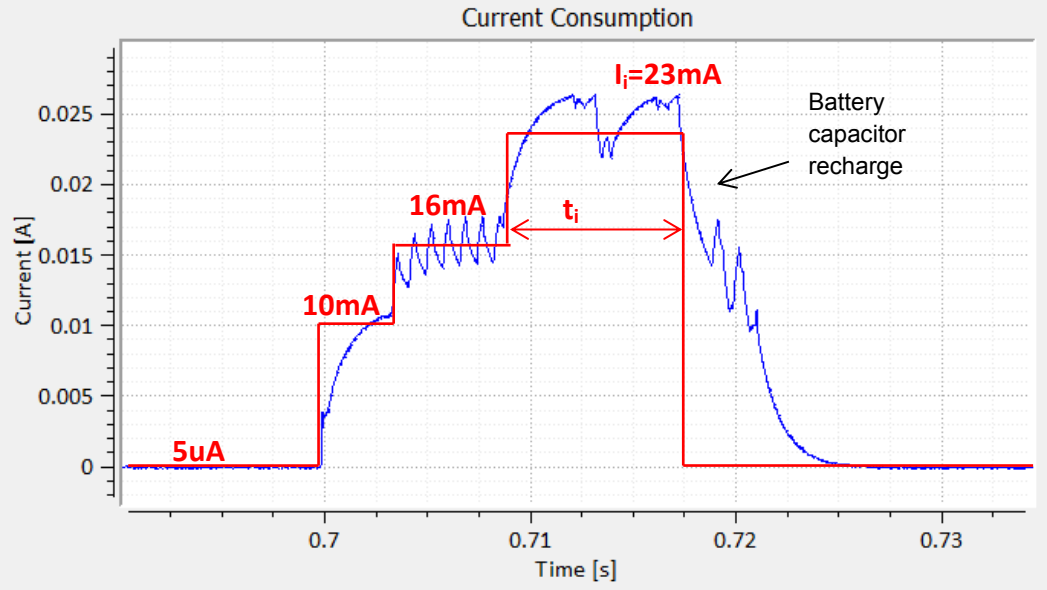

Figure 42. Average current consumption states during a radio transmission. The time the battery capacitor takes to recharge isn't taken in the calculation process. The energy spends to recharge the capacitor leads to less current drawn by the incoming process after the wake-ups.

The total charge Q consumed by the device can be requested by the user via a resource or enabled as a periodic notification to the application via an observer. Figure 43 shows the charge consumed during 30h of activity while performing sensors measurement with the following intervals: temperature, humidity, illuminance at 5 minutes interval, pressure measurement at 10 minutes interval and battery voltage with solar panel voltage/current at 2 minutes interval. The motion detector is enabled and performs detection every 10 minutes in average. The charge consumed by the system is reported to the application every 2 minutes. After 30h, the charge calculated by the online estimation algorithm is 4 coulombs. The slope of the curve gives us an average current of 37µA.

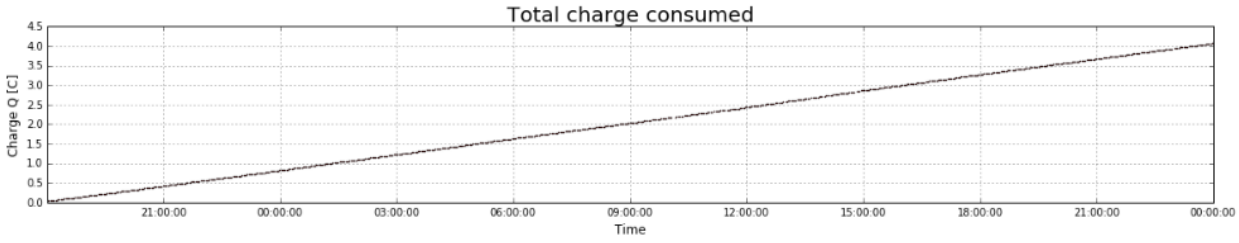

Figure 43. Online estimation of the charge consumed by the device with the Contiki Energest module.

To verify the computation of the consumed charge above, we calculated theoretically the charge by using the measured charges and durations from Table 11. Multiplying the charge of each task by its number of occurrences and summing all together we found a theoretical charge of 4.15C (Table 13) which is extremely close to the charge provided by the online estimation algorithm in Figure 43.

Task	Interval	Nb. occurrences	Charge	Total Charge
Temperature measure	300s	360	324 µC	117 mC
Humidity measure	300s	360	296 µC	107 mC
Illuminance measure	300s	360	384 µC	138 mC
Pressure measure	600s	180	340 µC	61 mC
Battery voltage measure	120s	900	359 µC	323 mC
Solar voltage measure	120s	900	359 µC	323 mC
Solar current measure	120s	900	266 µC	239 mC
Presence detection	600s (avg)	180	240 µC	43 mC
Energest reporting	120s	900	290 µC	261 mC
ARO Registration	3600s	30	371 µC	11 mC
LwM2M Registration Update	300s	360	5 mC	1.8 C
Standby	-	-	7µA x (30h-D)	720 mC
			TOTAL	**4.15 C**

Table 13. Total consumed charge over 30h calculated from values of Table 11 with D the sum all task's durations.

We realized this experiment several times for a total of about 230 hours and find an average difference of less than 10% between the theoretical charge and that reported by the online estimation algorithm (Table 14). The difference is due to the CoAP messages retransmission occurring sometimes, because of some small variation in the charge consumed by each task, and of course because of the errors introduced by the fixed average current configured in the software. However, the error remains small and provides a good indication of the energy consumption over the long term and can be a valuable alternative to using sophisticated measurement hardware [10].

Tasks	Duration [s]	Measured charge [C]	Theoretical charge [C]	Difference [%]
1	58740	10.48	10.3	1.73%
2	29840	32	27.14	16.44%
3	69660	9	9.61	6.56%
4	162300	20	22.4	11.32%
5	51600	55	54	1.83%
6	210600	31	28.88	7.08%
7	232620	30	31.9	6.14%
	815360		**Average Δ:**	**7.30%**
	226.49 hours			

Table 14. Percentage of difference between the theoretical charge and that measured by the online estimation algorithm.

System overview

We're using the border-router from the 6LBR project (https://github.com/cetic/6lbr) in the 6LOWPAN-RPL-ROUTER mode which enables the optimized ND and RPL protocols. The border-router can be installed on different kind of machine and it is recommended to follows the installation procedure provided in the documentation. The LwM2M Server can run on the same machine as the border-router providing that NodeJS is installed and the machine is connected to the same network (Figure 44).

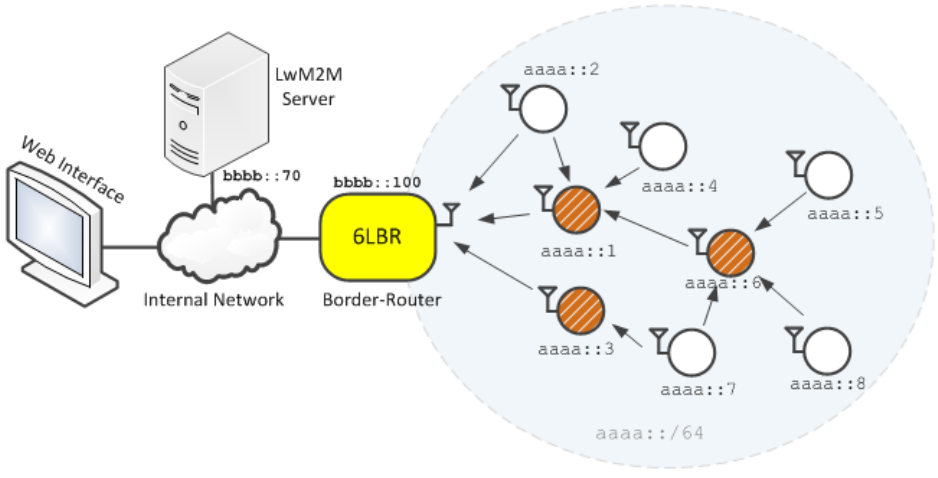

Figure 44. System overview.

References

[1] *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, RFC6775.

[2] M. A. M. Seliem, K. M. F. Elsayed, and A. Khattab, "Performance evaluation and optimization of neighbor discovery implementation over Contiki OS," in *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, 2015, pp. 119–123.

[3] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002, pp. 1567–1576.

[4] C.-J. Fu, A.-H. Lee, M.-H. Jin, and C.-Y. Kao, "A latency MAC protocol for wireless sensor networks," *International Journal of Future Generation Communication and Networking*, vol. 2, no. 1, pp. 41–54, 2009.

[5] "An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 224.

[6] *Neighbor Discovery for IP version 6 (IPv6)*, RFC4861, 2007.

[7] *Constrained Application Protocol (CoAP)*, RFC7252, 2014.

[8] *Lightweight M2M Specification v1.0*, 2014.

[9] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proceedings of the 4th workshop on Embedded networked sensors*, 2007, pp. 28–32.

[10] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder, "On the accuracy of software-based energy estimation techniques," in *Wireless Sensor Networks*: Springer, 2011, pp. 49–64.

Revision History

Revision	Author	Date	Comment
1.0	Darko Petrovic	26.04.2016	First release
0.1	Darko Petrovic	24.03.2016	Draft