



S5 Project  
9 March 2017  
Rennes, France.

# WIRELESS SENSOR NETWORK: CONTROL AND POWER MONITORING

BECERRA ESLAVA Gabriel Jaime  
GONZÁLEZ BARRETO Miguel Angel  
COSTA EPISCOPO Florencia



**IMT Atlantique**  
Bretagne-Pays de la Loire  
École Mines-Télécom



## INDEX

INTRODUCTION.....	4
THEORETICAL BACKGROUND.....	5
<b>WIRELESS SENSOR NETWORK</b> .....	5
<b>CONTIKI</b> .....	5
<b>PROTOCOLS</b> .....	5
<b>IEEE 802.15.4</b> .....	5
<b>SLIP</b> .....	5
<b>RPL</b> .....	5
<b>IPV6</b> .....	6
<b>CoAP</b> .....	6
<b>PROTOCOL STACK</b> .....	6
<b>EMONCMS</b> .....	6
<b>HOMADEUS SMARTPLUG</b> .....	7
<b>TMOTE-SKY</b> .....	7
MATERIALS.....	8
ARCHITECTURE.....	9
<b>PHYSICAL</b> .....	9
<b>LINUX COMPUTER</b> .....	9
<b>RASPBERRY PI</b> .....	10
<b>ACCES POINT</b> .....	10
<b>SWITCH</b> .....	10
<b>TMOTE SKY</b> .....	10
<b>HOMADEUS SMARTPLUGS</b> .....	10
<b>SMARTPHONE</b> .....	10
<b>LOGICAL</b> .....	10
INSTALATION GUIDE.....	12
<b>CLONE THE REPOSITORY</b> .....	12
<b>RASPBERRY PI 3 CONFIGURATION</b> .....	12
<b>COMPUTER CONFIGURATION</b> .....	14
<b>EMONCMS</b> .....	14
<b>HTTP – CoAP TRANSLATORS</b> .....	19
<b>HOMADEUS SMARTPLUG</b> .....	20
<b>SMARTPHONE CONFIGURATION</b> .....	22
DEPLOYING THE ARCHITECTURE.....	23
ANALYSIS AND FUTURE IMPROVEMENTS.....	26
DEMO.....	28
<b>ANNEXE</b> .....	29
<b>CODES</b> .....	29
<b>COAPCLIENT_HTTPCLIENT.PY</b> .....	29
<b>COAPCLIENT_HTTPSERVER.PY</b> .....	32

## INTRODUCTION

The internet of things is a new paradigm where we have “things” that can obtain different types of information and communicate it using various protocols and networks topologies.

Things can obtain data as: pollutions levels, humidity, temperature, energy consumption, among others. This data can be studied and used to monitor and have a bigger control of the applications where they are used.

Having the opportunity to deploy these things using wireless technologies is a huge advantage for the development of new applications since we do not have to worry about building an important physical infrastructure.

The first goal of this project is to build a wireless sensor network of smartplugs which can measure the energy consumption of an appliance and send it through the network to a remote data base using IEEE 802.15.4, 6LoWPAN, IPv6, RPL, UDP and CoAP protocols. The data received and stored should be shown in a graphic in order to see the evolution of the energy consumption.

The second goal is to control the WSN’s devices remotely, in this case the relay (actuator) of each smartplug which is responsible of switching on/off the appliance.

In order to achieve these objectives, in the next sections we will give a theoretical background of the most important protocols and technologies involved. We will also provide the materials used in this project, how the architecture is proposed and how the system works. Then, there will be a section dedicated to the installation of all the software and packages needed and also another one where we show the setting up of the architecture. After we do an analysis of the work done and results and finally we will describe a demo of this application. There is also an annexe with additional information.

## THEORETICAL BACKGROUND

### WIRELESS SENSOR NETWORK

Wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to monitor and record the physical conditions of the environment and organizing the collected data at a central location.

The devices usually have severe resource constraints in terms of energy, processing power and memory.

The wireless protocol you select to allow the communication between devices depends on your application requirements. Some of the available standards include 2.4 GHz radios based on either IEEE 802.15.4 or IEEE 802.11 (Wi-Fi) standards or proprietary radios, which are usually 900 MHz. In this project we will use IEEE 802.15.4.

### CONTIKI

Contiki is an open source operating system for the Internet of Things. It connects tiny low-cost, low-power microcontrollers to the Internet. Contiki is a powerful toolbox for building complex wireless systems.

It provides powerful low-power Internet communication supporting fully standard IPv6 and IPv4, UDP, TCP, and HTTP, along with the recent low-power wireless standards: 6LoWPAN, RPL, CoAP. With Contiki's ContikiMAC and sleepy routers, even wireless routers can be battery-operated.

Contiki applications are written in standard C, with the Cooja simulator Contiki networks can be emulated before burned into hardware which makes it easy to develop.

For the project we are going to install Contiki in all our WSN devices.

### PROTOCOLS

---

#### IEEE 802.15.4

IEEE 802.15.4 is a technical standard which specifies the physical layer and media access control for LR-WPANs. It intends to offer the fundamental lower network layers which focuses on low-cost, low-speed ubiquitous communication between devices.

---

#### SLIP

Serial Line Internet Protocol or SLIP is TCP/IP protocol used for communication between two machines that are previously configured for communication with each other, designed to work over serial ports and modem connections. SLIP defines a sequence of characters that frame IP packets on a serial line, and nothing more. It is applied in microcontrollers due to its very small overhead. Contiki uses SLIP to bridge the wireless IPv6 network onto a PC via a USB connection. Nowadays, SLIP has been largely supplanted by PPP (Point-To-Point Protocol), but Contiki uses it for establish communication between the constrained device and a regular Linux PC.

---

#### RPL

To accommodate the communication of IPv6 network, IETF comes up with Routing Protocol for Low-Power and Lossy Networks or RPL which provides the efficient paths for MP2P (MultiPoint-

To-Point) and P2MP (Point-To-MultiPoint) traffic patterns in LLNs. It implements measures to reduce energy consumption such as dynamic sending rate of control messages and addressing topology inconsistencies only when data packets have to be sent. One important thing, RPL isn't purposed to connect 6LoWPAN to router, border router is.

Border router (rpl-border-router Contiki implementation) or edge router is the router residing at the edge or boundary of a network. This router ensures the connectivity of its network with external networks, a wide area network or the Internet.

---

## IPV6

IPv6 is the successor of the current Internet Protocol IPv4 and addresses most limitations of IPv4 with large address space of  $2^{128}$ . IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard proposes a solution to use the IP on sensor nodes and to integrate these low-power devices into the Internet that benefits WSN which is conceded would be globally implemented however with constrained memory space.

---

## COAP

Constrained Application Protocol is an application protocol intended to be used in very simple electronics devices that allows them to communicate interactively over the Internet. Based on UDP, it provides a request/response interaction model, similar to the client/server model of HTTP between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. Request sent by client for an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation. CoAP is really ideal for constrained devices and networks.

## PROTOCOL STACK

The next table shows the protocol stack used in our WSN.

Layer	Protocol
Application	<b>CoAP</b>
Transport	<b>UDP</b>
Network	<b>IPv6 / RPL</b>
Adaption	<b>6LoWPAN</b>
MAC	<b>CSMA</b>
Radio Duty Cycling	<b>ContikiMAC</b>
Physical	<b>IEEE 802.15.4 (PHY)</b>

## EMONCMS

Emoncms is a powerful open-source web-app for processing, logging and visualising energy, temperature and other environmental data and is part of the OpenEnergyMonitor project.

There are two "types" of emoncms that we can use, one who runs locally (the standard version) and another one who runs in the cloud (on emoncms.org). There are some differences between

them. The emoncms.org version Github: emoncms/emoncmsorg is a fork that is specific for multi-server installations. While both versions share the same roots the code for emoncms.org differs significantly to the standard version of emoncms, the user experience is intended to be similar but there are currently a number of differences in the API and look of the inputs and feeds interfaces as well as a reduced feature set in general on emoncms.org in order to ensure stability. In general development on emoncms.org moves slower than the standard emoncms for this reason.

In the project we will use the standard version (version 9) who runs locally. Further information:

- <https://github.com/emoncms/emoncms>
- <https://emoncms.org/>

### **HOMADEUS SMARTPLUG**

Homadeus is the name of a smartplug developed in a previous university project. Homadeus is based on an AVR microcontroller and it runs Contiki. It works with IEEE 802.15.4 wireless protocol. We will use the smartplugs as the nodes of our WSN.

Further information:

- Tutorial - Installing a Homadeus smart power socket

### **TMOTE-SKY**

The Tmote-sky is an ultra low power IEEE 802.15.4 compliant wireless sensor module for use in sensor networks, monitoring applications, and rapid application prototyping.

Tmote Sky leverages industry standards like USB and IEEE 802.15.4 to interoperate seamlessly with other devices. By using industry standards, integrating humidity, temperature, and light sensors, and providing flexible interconnection with peripherals, Tmote Sky enables a wide range of mesh network applications.

Tmote Sky is a drop-in replacement for Moteiv's successful Telos design. Tmote Sky includes increased performance, functionality, and expansion. With TinyOS support out-of-the-box, Tmote leverages emerging wireless protocols and the open source software movement. Tmote Sky is part of a line of modules featuring on-board sensors to increase robustness while decreasing cost and package size.

The Tmote will be used as the border router of our WSN.

## MATERIALS

1. 1 Linux computer
2. 1 Raspberry Pi 3b
3. 1 Tmote sky
4. Homadeus smartplugs
5. 1 Aten master view kvme cs-1774 switch
6. 1 Linksys wireless access point
7. 3 Ethernet cables RJ-45



# ARCHITECTURE

## PHYSICAL

The deployed architecture must allow us to achieve the main goals of the project:

1. Measure the power consumption of an appliance connected to the smartplug and stock that information.
2. Control the state of the smartplug, switching it on or off remotely

In order to describe how this two functionalities are implemented, we will start by precisizing how the different components listed in the previous section are interconnected. As Fig.1 shows, the Linux computer, the Raspberry Pi 3b and the Linksys wireless access point are all connected via the Ethernet cables to the switch. Then, the Tmote sky is branched to one of the USB ports of the Raspberry Pi, establishing a SLIP connection. Finally, the smartphone communicates wirelessly with the access point via WiFi and the Homadeus smartplugs communicates wirelessly with the Tmote sky via IEEE 802.15.4. The configuration need to be done in each of these components in order to have the desired behavior will be explained in the next section of this document.

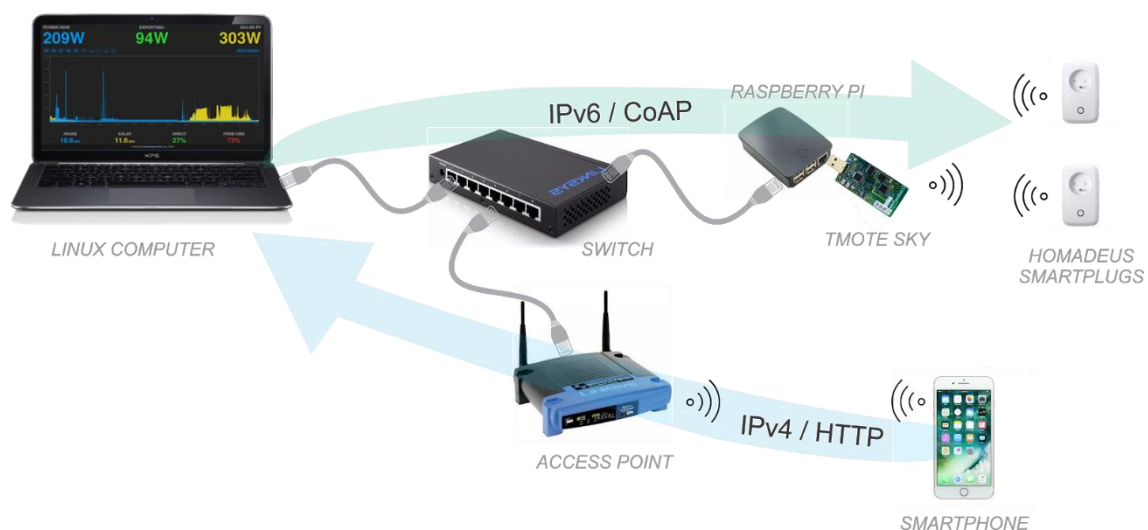


Fig. 1: Global architecture

Now that the interconnection of all the components has been established, it's time to precise their role in the current architecture.

### LINUX COMPUTER

The computer will host a local database and an Emoncms instance, both necessary in order to stock and visualize the received information about the power consumption of the smartplugs. The database will stock the received data and the Emoncms WebApp will represent this data as a graphic in the computer screen.

Additionally, the computer will host both a CoAP-HTTP translator and an HTTP-CoAP translator, which have been developed in the context of this project in order to accomplish both the power consumption measurement and the actuator of the smartplugs. Having said this, the computer can act as a CoAP client, an HTTP client and an HTTP Server depending on the context in which

it is being used. We will explain in details how these functions are accomplished as well as the implementation of the translators later in this document.

---

## **RASPBERRY PI**

The Raspberry Pi acts as an IPv6 router, it receives the packets originated in the Linux computer and forwards them inside the RPL network. Equivalently, it forwards the responses originated in the SmartPlugs back to the computer.

---

## **ACCES POINT**

The access point has been introduced to the architecture in order to allow wireless devices like a Smartphone to have wireless access to the network. It also assigns IPv4 addresses to the others devices connected to the switch.

---

## **SWITCH**

The switch allows us to constitute a Local Area Network with the computer, the Raspberry Pi, the access point and other potential wireless devices presents (smartphones).

---

## **TMOTE SKY**

The Tmote Sky constitutes a RPL border router, and then the entry point to the Wireless Sensor Network of smartplugs.

---

## **HOMADEUS SMARTPLUGS**

The Homadeus smartplugs are capable of measuring the power consumption and they can be actionated remotely. They actually constitute CoAP servers, which receive CoAP requests originated in the computer and respond to them, with CoAP responses.

---

## **SMARTPHONE**

The smartphone constitutes an HTTP client, capable of sending an HTTP request in order to switch on or off the smartplug. This request will be traduced in a CoAP request that will be sent to the Homadeus smartplug.

---

## **LOGICAL**

We can divide the logical architecture in two parts.

The first one is in charge of the power consumption measurements communication between the smartplug (CoAP server) and the emoncms data base (HTTP server).

The second part is in charge of the switching on/off communication between the remote control (HTTP client) and the smartplug (CoAP server).

To reach these communications we needed to implement 2 CoAP-HTTP translators, the first one (CoapClient\_httpClient.py) to communicate with the CoAP server and the HTTP server and a second one (coapClient\_httpServer.py) to communicate with CoAP server and an HTTP client.

---

## **COAPCLIENT\_HTTPCLIENT.PY**

This script allows us to implement the functionality of reading the power consumption of the smartplug and stocking it in the database. It is a Python implementation of a CoAP client and an HTTP client.

First, it asks the Homadeus for the power consumption by sending a CoAP GET request to the smartplug (1). The Homadeus send back a CoAP response with the power consumption information (2), which coapClient\_httpClient.py will traduce in an HTTP POST request. It will then send the HTTP POST request to the Emoncms (3), who will store it in the database and show it in a graphic in the WebApp.

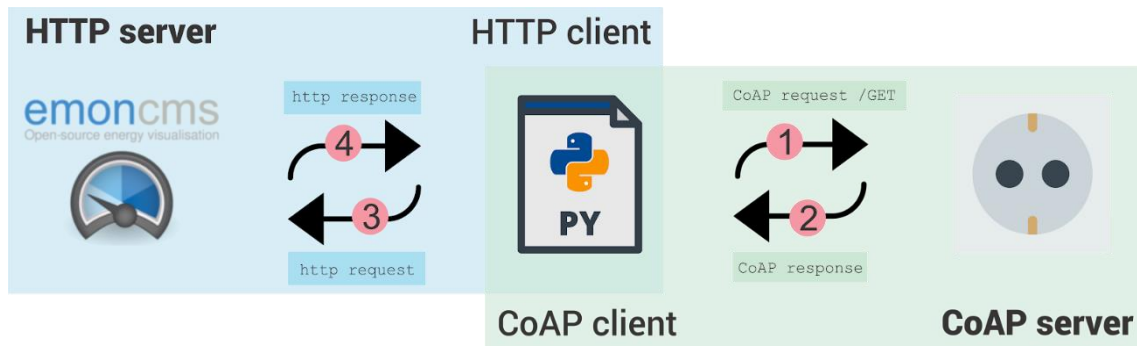


Fig. 2: coapClient\_httpClient.py

#### COAPCLIENT\_HTTPSERVER.PY

This script allows us to implement the functionality of switching on and off the smartplug remotely. It is a Python implementation of an HTTP server and a CoAP client.

In this case, the action is initiated by an HTTP client, which in Figure 3 is implemented with Emoncms via the curl module of its Dashboard. However, this can be done by any HTTP client, for example, a smartphone that sends an HTTP GET request to our HTTP server, as we will see in the Demo section.

First, the Emoncms sends an HTTP POST request to the HTTP server implemented in `coapClient_httpServer.py` (1). Then, `coapClient_httpServer.py` will traduce this in a CoAP POST request and send it to the smartplug (3), which will change its state from ON to OFF or vice versa.

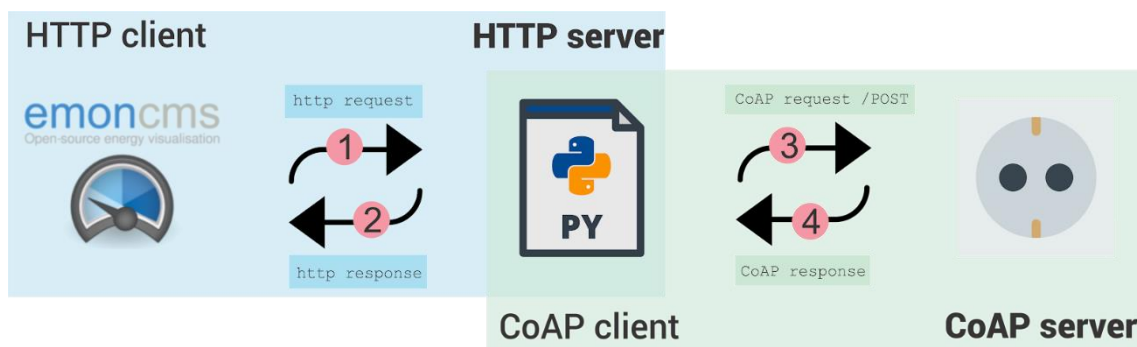


Fig. 3: coapClient\_httpServer.py

## INSTALLATION GUIDE

Before deploying the architecture presented in the previous section, there are a series of steps need to be done in order to set up the different devices.

### CLONE THE REPOSITORY

The first step is cloning the repository from: <https://redmine-df.telecom-bretagne.eu/git/wsns5> to the Linux Computer. Here you will find the Raspbian image used with all configurations and installations already done (ready to flash in the Pi), the smartplug libraries in order to configure the Homadeus as well as all the scripts used in this project.

```
git clone https://redmine-df.telecom-bretagne.eu/git/wsns5
```

### RASPBERRY PI 3 CONFIGURATION

The first three steps of this section is to configure the Raspberry Pi from zero in order to reach a successful ssh connection from another computer, so if you already can do this, you can just skip these three steps.

#### 1) Raspbian installation

- a) SD card format
  - i) In order to ensure your SD card is completely blank and has no partitions it is recommendable to format it with an SD formatter program like SD Card Formatter or similar.
- b) Raspbian download
  - i) Download the "Raspbian Stretch with Desktop" zip
- c) Writing an image to the SD card
  - i) You will need to use an image writing tool to install the image (zip) you have downloaded, on your SD card.
  - ii) Download Etcher and install it.
  - iii) Connect the SD card.
  - iv) Open Etcher and select from your hard drive the Raspberry Pi .img or .zip file.
  - v) Select the SD card you wish to write your image to.
  - vi) Review your selections and click 'Flash!' to begin writing data to the SD card.

#### 2) Enable SSH on a headless Raspberry Pi

- a) SSH can be enabled by placing a file named ssh, without any extension, onto the boot partition of the SD card from another computer. When the Pi boots, it looks for the ssh file. If it is found, SSH is enabled and the file is deleted. The content of the file does not matter; it could contain text, or nothing at all.
- b) If you have loaded Raspbian onto a blank SD card, you will have two partitions. The first one, which is the smaller one, is the boot partition. Place the file into this one.

#### 3) First SSH connection

- a) There are many ways to assign an IP to the Raspberry Pi in order to create a SSH connection. We are going to explain one.
  - i) Connect the Pi to an access point (router) with Internet connection through an Ethernet cable.
  - ii) From another computer connected via Ethernet cable to the same access point execute wireshark and look for the IP address of the Pi (from now on we call it pi\_ip\_address).
  - iii) Open the terminal in the computer and execute:

```
ssh pi@pi_ip_address
```

The password is raspberrry

#### 4) Set up CONTIKI

Once we are in the Raspberry through the ssh connection and it is connected to the internet via the access point we are going to install Contiki executing:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

Cloning repository and updating dependencies

```
git clone https://github.com/contiki-os/contiki.git
cd contiki
git submodule sync && git submodule update --init
git pull
```

Installing dependencies

```
sudo apt-get install build-essential libncursesw5-dev libgdbm-dev libc6-dev
zlib1g-dev libsqlite3-dev tk-dev libssl-dev openssl
sudo apt-get install gcc-msp430
sudo apt-get install gcc flex bison libboost-graph-dev
sudo apt-get install build-essential binutils-msp430 gcc-msp430 msp430-libc
msp430mcu
sudo apt-get install mspdebug binutils-avr gcc-avr gdb-avr avr-libc avrdude
openjdk-7-jdk openjdk-7-jre ant libncurses5-dev bridge-utils build-essential
binutils-msp430 gcc-msp430 msp430-libc msp430mcu
```

Enabling IPv6

```
sudo modprobe ipv6
sudo /bin/bash // Pour passer en root
sudo echo "ipv6" >> /etc/modules
sudo echo "iface eth0 inet6 dhcp" >> /etc/network/interfaces
```

#### 5) IPv6 configuration

- a) In order to communicate with the Pi using an Ethernet cable directly from the computer we will assign a static IP address.
- b) Open a terminal in the Pi, go to /etc on the root directory and execute:

```
sudo nano dhcpd.conf
```

and uncomment the "Exemple static IP Configuration" lines as follows:

```
interface eth0
static ip_address=192.168.0.10/24
static ip6_address=fd01::212:7400:13e2:dd10/64
static routers=192.168.0.1
static domain_name_server=192.168.0.1 8.8.8.8 fd01::212:7400:13e2:1
```

These IP addresses are just an example, the important idea is to create two different network prefixes in IPv6 with the aim of using the Pi as a router.

- c) Ctrl+X to exit and Enter (to save changes)

## 6) Enable port forwarding

a) Go to /etc on the root directory and execute:

```
sudo nano sysctl.conf
```

Uncomment the next lines:

```
net.ipv4.ip_forward=1
```

```
net.ipv6.conf.all.forwarding=1
```

b) Ctrl+X to exit and Enter (to save changes)

```
sudo reboot
```

## COMPUTER CONFIGURATION

### EMONCMS

#### INSTALLATION

#### RUNNING WITH INTERNET CONNECTION

In this case it is not necessary to install anything.

Go to <https://emoncms.org/> and create an account by entering your email, a username and a password and clicking register to complete.

#### RUNNING WITHOUT INTERNET CONNECTION

We are going to install Emoncms in the computer in order to save the data in a local database and be able to run the monitoring system without internet connection.

We open the terminal in the computer and follow next steps.

#### INSTALL DEPENDENCIES

*Update the system repositories.*

```
sudo apt-get update
```

As it is running in Ubuntu 14.04:

```
sudo apt-get install apache2 mysql-server mysql-client php5 libapache2-mod-php5 php5-mysql php5-curl php-pear php5-dev php5-mcrypt php5-json git-core redis-server build-essential ufw ntp -y
```

Set a password for the MySQL user

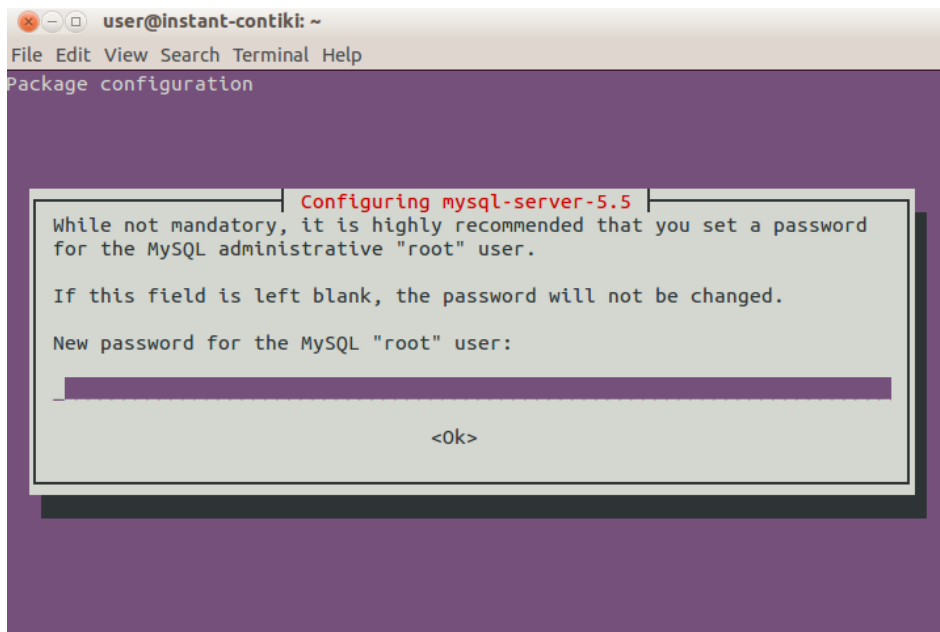


Fig. 4: MySQL

#### *Install PHP pecl dependencies*

Not essential, required for mail sending e.g. password recovery

```
sudo pear channel-discover pear.swiftmailer.org
sudo pecl install swift/swift dio-0.0.9 redis
```

enable igbinary serializer support? [no] : no

#### *Add pecl modules to php5 config*

```
sudo sh -c 'echo "extension=dio.so" > /etc/php5/apache2/conf.d/20-dio.ini'
sudo sh -c 'echo "extension=dio.so" > /etc/php5/cli/conf.d/20-dio.ini'
sudo sh -c 'echo "extension=redis.so" > /etc/php5/apache2/conf.d/20-redis.ini'
sudo sh -c 'echo "extension=redis.so" > /etc/php5/cli/conf.d/20-redis.ini'
```

#### *Configure Apache*

Emoncms uses a front controller to route requests, mode rewrite needs to be configured:

```
sudo a2enmod rewrite
sudo sh -c "echo '<Directory /var/www/html/emoncms>' >> /etc/apache2/sites-available/emoncms.conf"
sudo sh -c "echo ' Options FollowSymLinks' >> /etc/apache2/sites-available/emoncms.conf"
sudo sh -c "echo ' AllowOverride All' >> /etc/apache2/sites-available/emoncms.conf"
sudo sh -c "echo ' DirectoryIndex index.php' >> /etc/apache2/sites-available/emoncms.conf"
sudo sh -c "echo ' Order allow,deny' >> /etc/apache2/sites-available/emoncms.conf"
```

```

sudo sh -c "echo ' Allow from all' >> /etc/apache2/sites-
available/emoncms.conf"
sudo sh -c "echo '</Directory>' >> /etc/apache2/sites-
available/emoncms.conf"
sudo sh -c "echo 'ServerName localhost' >> /etc/apache2/apache2.conf"
sudo ln -s /etc/apache2/sites-available/emoncms.conf /etc/apache2/sites-
enabled/
sudo a2ensite emoncms
sudo service apache2 reload

```

## INSTALL EMONCMS

First cd into the /var/www directory

```
cd /var/www/
```

Set the permissions of the html directory to be owned by your username (user in Instant Contiki virtual machine)

```
sudo chown user html
```

Cd into html directory

```
cd html
```

Download emoncms using git

```
git clone -b stable https://github.com/emoncms/emoncms.git
```

Once installed you can pull in updates with

```
cd /var/www/html/emoncms
git pull

```

## CREATE A MYSQL DATABASE

```
mysql -u root -p
```

Enter the MySQL password that you set above. Then enter the sql to create a database:

```
mysql> CREATE DATABASE emoncms DEFAULT CHARACTER SET utf8;
```

Then add a user and password to the database for emoncms.

```
mysql> CREATE USER 'emoncms_db_username_here'@'localhost' IDENTIFIED BY
'emoncms_db_password_here';
```

Give it permissions on the new database.

```
mysql> GRANT ALL ON emoncms.* TO 'emoncms_db_username_here'@'localhost';
```

```
mysql> flush privileges;
```

Exit MySQL by:

```
mysql> exit
```

## CREATE DATA REPOSITORIES FOR EMONCMS FEED ENGINES

```
sudo mkdir /var/lib/phpfiwa
```



```
sudo mkdir /var/lib/phpfina
sudo mkdir /var/lib/phptimeseries

sudo chown www-data:root /var/lib/phpfiwa
sudo chown www-data:root /var/lib/phpfina
sudo chown www-data:root /var/lib/phptimeseries
```

## SETUP EMONCMS SETTINGS

---

cd into the emoncms directory where the settings file is located

```
cd /var/www/html/emoncms/
```

Make a copy of default.settings.php and call it settings.php

```
cp default.settings.php settings.php
```

Open settings.php in an editor:

```
nano settings.php
```

Update your database settings to use your new user and secure password:

```
$server = "localhost";
$database = "emoncms";
$username = "emoncms_db_username_here";
$password = "emoncms_db_password_here";
$port = "3306";
```

You will also want to modify SMTP settings and the password reset flag further down in the settings file.

```
// Allow user to reset his password
$ enable_password_reset = true;

// (OPTIONAL) Email SMTP, used for password reset or other email functions
$ smtp_email_settings = array(
    'host'=>"smtp.gmail.com",
    'port'=>"465", // 25, 465, 587
    'from'=>array('noreply@emoncms.org' => 'EmonCMS'),
    // comment lines below that dont apply
    'encryption'=>"ssl", // ssl, tls
    'username'=>"yourusername@gmail.com",
    'password'=>"emoncms_db_password_here "
);
```

Save (Ctrl-X), type Y and exit

## INSTALL ADD-ON EMONCMS MODULES (OPTIONAL)

---

```
cd /var/www/html/emoncms/Modules
git clone https://github.com/emoncms/dashboard.git
git clone https://github.com/emoncms/app.git
```

The 'modules' need to save their configurations in the emoncms database, so once you run emoncms (next step) in your browser - update your emoncms database: Setup > Administration > Update database.

## RUNNING EMONCMS

---

Go to <http://localhost/emoncms>

The first time you run emoncms it will automatically setup the database and you will be taken straight to the register/login screen.

Create an account by entering your email, a username and a password and clicking register to complete. The username and password don't need to be the same as the emoncms\_db we have created before

### *Configure PHP Timezone*

PHP 5.4.0 has removed the timezone guessing algorithm and now defaults the timezone to "UTC" on some distros. To resolve this:

Open php.ini

```
sudo nano /etc/php5/apache2/php.ini
```

and search for "date.timezone" with ctrl+w

```
[Date]
; Defines the default timezone used by the date functions.
; http://php.net/date.timezone
;date.timezone =
```

edit date.timezone to your appropriate timezone:

```
date.timezone = "Europe/Amsterdam"
```

(PHP supported timezones are listed here: <http://php.net/manual/en/timezones.php>)

Now save and close (Ctrl + x) and restart your apache.

```
sudo service apache2 restart
```

### *Install Logger*

Update your emoncms installation to ensure that the necessary files are downloaded:

```
cd /var/www/html/emoncms && git pull
```

Make the installation script executable:

```
cd /var/www/html/emoncms/scripts/logger/ && sudo chmod +x install.sh
```

Run the installation script:

```
sudo ./install.sh
```

Provided that the directory /var/log has been successfully mounted in tmpfs, a **system reboot is necessary to complete the process.**

## ADDING THE REMOTE CONTROL

---

In order to switch on/off the smartplugs remotely we need to configure a dashboard on emoncms who send the request.

- 1) Go to <http://localhost/emoncms>
- 2) Login to your account
- 3) On the upper right hand corner Setup -> Dashboards -> new
- 4) Clic on the edition button of the created dashboard
- 5) It will appear a work area with a toolbox, clic on the speed meter icon and select curl option
- 6) Clic on the work area, and once the curl box appears configure it with the configure button at the toolbox as is shown in the next image

The image shows a 'Configure element' dialog box with the following fields and values:

IP	127.0.0.1	IP address of server
Port	8081	Listen port of server
URL	id00::444f:4f50:a74b:2828/5683/	URL example: node/param
Payload		Data to send
Method	GET	GET/POST
HTTPS	no	yes/no
Timeout	100	in milliseconds
Colour	1	0=rd, 1=gn, 2=gy, 3=bu, 4=vio, 5=ye, >5=bk
Caption	Switch	Button Text
Confirmation	no	Confirmation Box: yes/no

Buttons: Cancel, Save changes

Fig. 5: Curl module

As we can see, in IP and port we write the address of the computer is running the HTTP-CoAP translator (explained in next step), in URL we write smartplug\_IP/smartplug\_port/ and GET as the method.

## HTTP – COAP TRANSLATORS

The actual version of Emoncms does not supports CoAP requests, it only supports HTTP, while the smartplug is a CoAP server. Given that we need both the Emoncms instance in the Linux Computer and the smartplug to exchange data, we need translators between HTTP and CoAP. That is what we implemented in the scripts `coapClient_httpClient.py` and `coapClient_httpServer.py`, that you have cloned to the Linux computer when cloning the repository. In order to run these scripts we need to install a CoAP client library.

### COAP CLIENT LIBRARY

Before we can use our CoAP client implementation in both Python scripts described above, we have to instal a library called CoAPthon. This library developed in python allows us to create CoAP requests to interrogate the Smartplug.

To install the library we do as follows:

```
sudo apt-get install python-pip
sudo pip install CoAPthon
```

## HOMADEUS SMARTPLUG

One of the most important components is the Homadeus Smartplug. This is the component that allows to measure the energy consumption but is also the actuator, we can switch it on and off in order to feed whatever is connected to the smartplug. To do that we need Homadeus socket and USBasp programmer packet which will be rolling to upload the code to microchip. In the Homadeus Smartplug, there is already a bundle of code to do measurement as the function of smart grid.

First we are going to see the hardware and the connexion and then we'll talk about the packages that should be installed in a computer in order to flash the application.

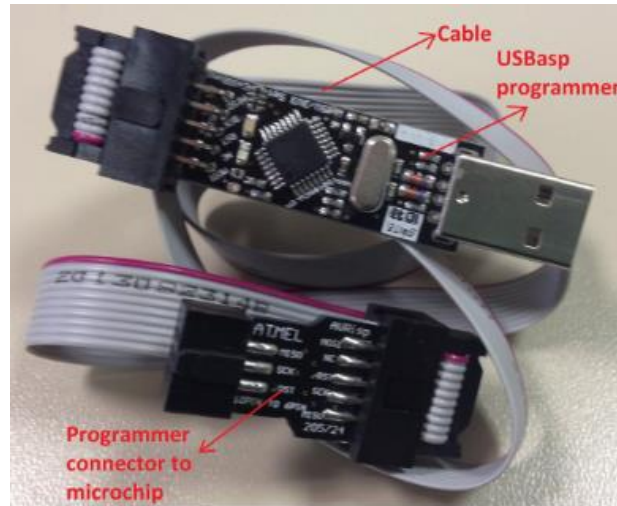


Fig. 6: USBasp AVR Programming Device for ATMEL processors



Fig. 7: The Homadeus Smartplug

They should be branched as follows:

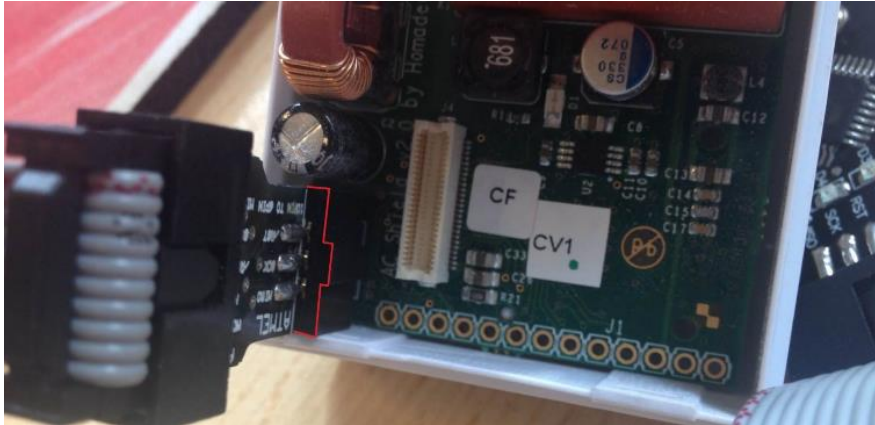


Fig. 8: Connection to flash the smartplug

The other end of the cable should be connected to the Linux computer. You should pay attention to the correct position of the cable, if you do not connect it as it is on the image you can damage the smartplug permanently

For the next part you can use any computer, we recommend you to keep using the Linux computer where you have been working on. Install the next packages before you upload the program to the Homadeus smartplug:

```
sudo apt-get install build-essential libncursesw5-dev libgdbm-dev / libc6-
dev zlib1g-dev libsqlite3-dev tk-dev libssl-dev openssl
sudo apt-get install gcc-msp430
sudo apt-get install gcc flex bison libboost-graph-dev
sudo apt-get install build-essential binutils-msp430 gcc-msp430 msp430-
libc msp430mcu
```

After having done this you should clone the repository: <https://redmine-df.telecom-bretagne.eu/git/wsns5> (if you work on another computer or if you have not clone it), or just copy the /smartplug folder. Once you have done this that you can upload the program to the smartplug ( which is a CoAP server)

```
cd wsns5/smartplug/demo/er-rest-homadeus
make TARGET=avr-homadeus
```

If you get this message:

```
INFO: compiling with CoAP-13
make: Nothing to be done for 'all'
```

Run the command 'make clean' and re-compile. Then upload the code by doing:

```
make avrdude
```

If it is successfully uploaded you should see this:

```
File Edit View Search Terminal Help
avrdude: 88236 bytes of flash written
avrdude: verifying flash memory against er-rest-homadeus.avr-homadeus.hex:
avrdude: load data flash data from input file er-rest-homadeus.avr-homadeus.hex:
avrdude: input file er-rest-homadeus.avr-homadeus.hex auto detected as Intel Hex
avrdude: input file er-rest-homadeus.avr-homadeus.hex contains 88236 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 30.81s

avrdude: verifying ...
avrdude: 88236 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done. Thank you.

rm er-rest-homadeus.avr-homadeus.out er-rest-homadeus.co er-rest-homadeus.avr-ho
madeus.hex
user@instant-contiki:~/homadeus/repo/demo/er-rest-homadeus$
```

Fig. 9: Successfully upload

Now when we run the application of the rpl border router in the Tmote Sky we should be able to establish a communication with the smartplug.

### SMARTPHONE CONFIGURATION

To use the smartphone as an HTTP client and be able to create the request needed to switch on/off the smartplugs we just need to download an app called *HTTP Shortcuts* for Android and install it. We have chosen this app because is very easy to configure and has a simple interface.

Once we have installed the app we have to follow the next steps in order to create a button capable of switch the smartplug.

- 1) We create a new shortcut touching the + icon, then we have to complete the following options:
- 2) Give a name to the shortcut
- 3) Request method : GET
- 4) URL: `http://192.168.1.100:8081/fd00::444f:4f50:a74b:2828/5683`
- 5) No authentication
- 6) Response type: Simple toast
- 7) Timeout: medium (10 sec)
- 8) No delay
- 9) Drop the request

The part 3. is an example, the IPv4 address is the linux computer IP assigned by the access point, the port is always 8081, and the IPv6 address is the smartplug we want to switch, 5683 is the smartplug port.

## DEPLOYING THE ARCHITECTURE

After having followed the installation guide we are able to deploy the whole architecture.

- 1) Connect the devices as shown at Fig. 1
- 2) New SSH IPv6 connection to the Pi
  - a) Assign a static IP address to the computer, this IP address should have the same prefix as the one assigned to the Pi in step 5.b. of Raspberry pi 3 configuration

```
sudo ip -6 addr add fd01::212:7400:13e2:dd09/64 dev eth0
```

The line above is just an example, eth0 in this case is the interface connected to the Pi (in some computers is eno0)

- b) Assign the Pi static IPv6 address as default gateway in the computer

```
sudo ip -6 route add default via fd01::212:7400:13e2:dd10
```

- c) Now we are ready to connect to the Pi directly using SSH.

```
ssh -6 pi@fd01::212:7400:13e2:dd10
```

The line above is just an example, we have to put the address we gave to the Pi in step 5.b. of Raspberry pi 3 configuration

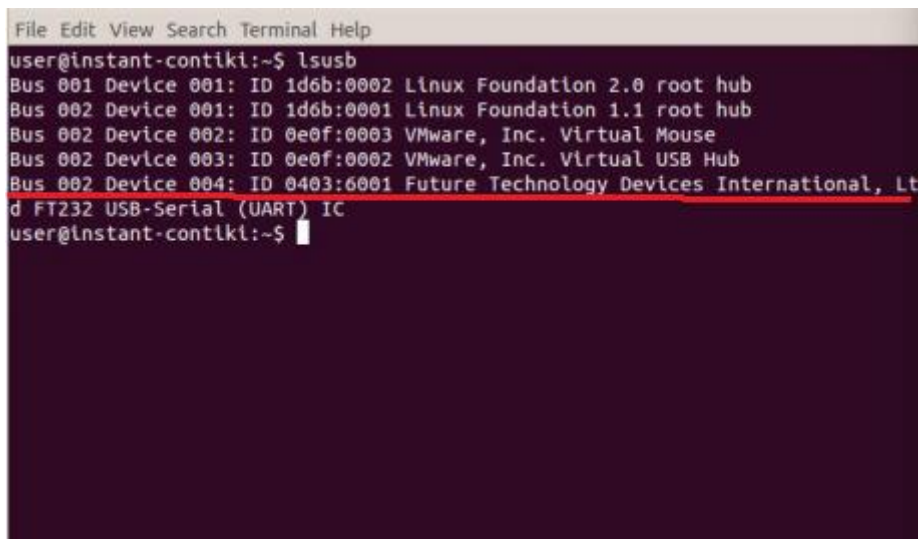
- d) The password is raspberry

- 3) Once we are in SSH connection with the Pi we need to create the SLIP connection with the tmote-sky to be able to communicate with the smartplug. We also have to configure the Tmote Sky in order to make it work as a RPL border router:

- a) Connect the Tmote Sky to an USB port in the computer and execute the next command to check if the hardware has been recognized:

```
lsusb
```

The result will be something like this:



```
File Edit View Search Terminal Help
user@instant-contiki:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 004: ID 0403:6001 Future Technology Devices International, Ltd
d FT232 USB-Serial (UART) IC
user@instant-contiki:~$
```

Sometimes there are permission errors in the port like “could not open port permission: “/dev/ttyUSB0”” That error means you are not allowed to access /dev/ttyS0 on your computer. Only root and users in the dialout group may access that device.

When such error happens, it may your user isn't in dialout group. In order to solve this issue, do:

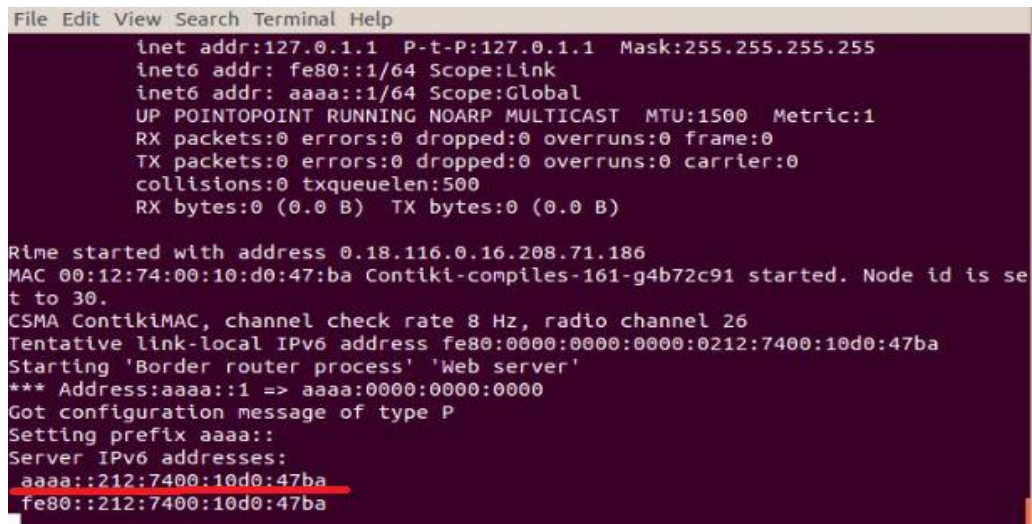
```
sudo adduser MyUser dialout
sudo chmod a+rw /dev/ttyUSB0
```

MyUser is your username. After doing this you can repeat the step a)

- 4) Upload the rpl-border-router to the tmote-sky and connect to it

```
cd contiki/examples/ipv6/rpl-border-router
make Target=sky savetarget
sudo make clean
make Target=sky border-router.upload MOTE=1
make connect-router
```

After following these steps you will find the IPv6 address of the Tmote sky border router application (sometimes you have to push the reset button on the Tmote-sky if the address is not shown, if it doesn't work you will probably have to re-connect the tmote again and start all the steps).



```
File Edit View Search Terminal Help
inet addr:127.0.1.1 P-t-P:127.0.1.1 Mask:255.255.255.255
inet6 addr: fe80::1/64 Scope:Link
inet6 addr: aaaa::1/64 Scope:Global
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Rime started with address 0.18.116.0.16.208.71.186
MAC 00:12:74:00:10:d0:47:ba Contiki-compiles-161-g4b72c91 started. Node id is set to 30.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7400:10d0:47ba
Starting 'Border router process' 'Web server'
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
  aaaa::212:7400:10d0:47ba
  fe80::212:7400:10d0:47ba
```

Fig. 10: tmote-sky IPv6

You can make a curl to this address to see the smartplugs that are present in the RPL network, just know that it may take some time to detect them.

```
curl http://[aaaa::212:7400:10d0:47ba]
```

- 5) In the linux computer run Emoncms :
  - a) Go to <http://localhost/emoncms> and login into your account.
  - b) Go to Setup at the upper right hand corner and copy your write API Key.
- 6) Open the coapClient\_httpClient.py script and paste your API key in the parameter API\_KEY, save. Open a new terminal and run the coapClient\_httpClient.py script.

```
python coapClient_httpClient.py -o GET -p
coap://[fd00::4444f:4f50:a74b:2828]:5683/pwr/w
```

At this moment the emoncms is storing the data of the smartplug, you can check it going in emoncms to setup -> inputs



7) Open a new terminal and run the coapClient\_httpServer.py script.

```
python coapClient_httpServer.py
```

## ANALYSIS AND FUTURE IMPROVEMENTS

At the beginning of the project we received a bunch of Homadeus Smartplugs, a pair of Tmote Sky devices and some tutorials explaining step by step how to configure both the Tmote Sky and the smartplug in order to exchange CoAP requests and responses. To do so we used Copper, a CoAP user-agent provided by Firefox.

The idea was then going further and deploy a network architecture that would enable this exchanges to happen otherwise, with a CoAP client running elsewhere rather than in the same device where the Tmote Sky was connected to. This would allow us to extend the scope of the system, and then think of multiple possible applications.

In this sense, we decided that the CoAP requests would be originated in a Linux computer, which would be able to reach the RPL Border Router (Tmote Sky) branched to a Raspberry Pi. The Raspberry then, would have to behave as an IPv6 router, and that is what we did.

We used Emoncms in the Linux computer to store and show the information of the power consumption, a tool that could be potentially used to analyse this data. The actuator was also implemented in Emoncms, through a Dashboard module; but also in smartphones attached to the network, by downloading an application able to send HTTP requests. We managed the conversion from HTTP to CoAP and viceversa through translators developed in Python.

One of the interests of the project was to study the hops between smartplugs in order to make a multi-hop WSN. We encountered some difficulties as the coverage range of the Tmote Sky is huge compared with that of the smartplugs. When running the RPL algorithm the RPL border router was always chosen as a father. Close to the end of the project we received some antennas to increase the coverage range of the smartplugs, but even when these antennas the chosen father was the RPL border router. We could not manage to create a multi-hop network of smartplugs in the time we had.

Despite we have reached the initial objectives, it's possible to make many improvements in terms of the logical architecture and functionalities of each part involved. We have thought about some improvements that could be made as part of a future continuation of the project.

Power consumption communication improvements: Instead of having a script who send GET requests to the smartplugs and then POST the information received to the emoncms database, it would be interesting to study others configurations. Following this line we propose to study the implementation of a CoAP client as a module of Emoncms itself that can GET power consumption information from the smartplug and save it in the database, in order to have a CoAP end-to-end communication, and to be able to add smartplugs power consumption from the emoncms directly instead of running different scripts. To optimise the previous idea we could set up the smartplugs as observable objects with the aim of reducing the GET request to it. It would mean changing the smartplug code, which may not be as easy as it seems.

Remote control communication improvements: Having as goal an end-to-end CoAP communication it would be interesting to add a CoAP client to Emoncms dashboard modules similar to the existing curl module, to be able to send POST CoAP requests directly to the smartplug from the Emoncms without the necessity of passing through an HTTP-CoAP translator.

IP configuration: Another point to improve is the IPv6 configuration, in this project we give manually a static IPv6 address to the Pi and to the Linux computer but it would be a good idea to give the addresses automatically implementing RADVD in the Pi.

## DEMO

The idea of this part of the demonstration presented at the FORUM, is to show the functionality of remote actuator implemented by the system. To do so, we have connected four different bulbs to four Homadeus smartplugs. We have deployed the already known architecture and we have configured 4 bottoms in the Smartphone application, each one of them associated with the IPv6 address of one smartplug. This way, when pressing one of the bottoms, an HTTP POST request will be sent to the HTTP server implemented by `coapClient_httpServer.py`, which runs in the Linux computer. There it will be traduced and sent to the corresponding Homadeus smarplug, as a CoAP POST request.

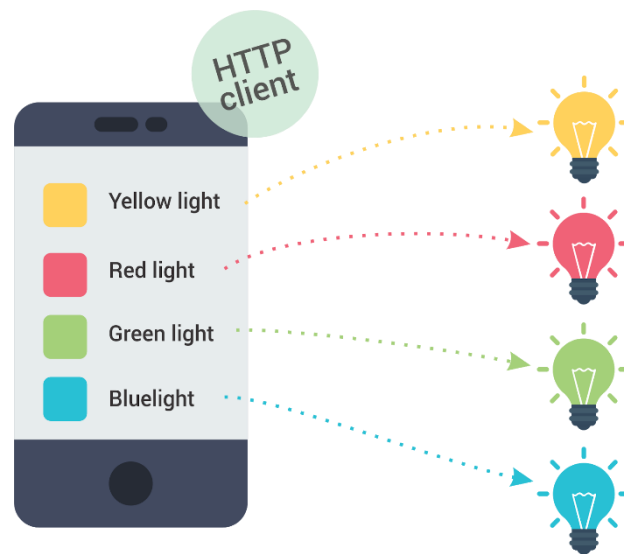


Fig. 11: Demo

# ANNEXE

## CODES

### COAPCLIENT\_HTTPCLIENT.PY

```
#!/usr/bin/env python
import getopt
import socket
import sys
import requests
import time
import random
import urllib
import json
from coapthon.client.helperclient import HelperClient
from coapthon.utils import parse_uri

client = None

def usage(): # pragma: no cover
    print "Command:\tcoapclient.py -o -p [-P]"
    print "Options:"
    print "\t-o, --operation=\tGET|PUT|POST|DELETE|DISCOVER|OBSERVE"
    print "\t-p, --path=\t\t\tPath of the request"
    print "\t-P, --payload=\t\tPayload of the request"
    print "\t-f, --payload-file=\t\tFile with payload of the request"

def client_callback(response):
    print "Callback"

def client_callback_observe(response): # pragma: no cover
    global client
    print "Callback_observe"
    check = True
    while check:
        chosen = raw_input("Stop observing? [y/N]: ")
        if chosen != "" and not (chosen == "n" or chosen == "N" or chosen ==
"y" or chosen == "Y"):
            print "Unrecognized choose."
            continue
        elif chosen == "y" or chosen == "Y":
            while True:
                rst = raw_input("Send RST message? [Y/n]: ")
                if rst != "" and not (rst == "n" or rst == "N" or rst == "y" or
rst == "Y"):
                    print "Unrecognized choose."
                    continue
                elif rst == "" or rst == "y" or rst == "Y":
                    client.cancel_observing(response, True)
                else:
                    client.cancel_observing(response, False)
            check = False
            break
        else:
            break
```

```

def main(): # pragma: no cover
    global client
    global payload
    op = None
    path = None
    payload = None

    try:
        opts, args = getopt.getopt(sys.argv[1:], "ho:p:P:f:", ["help",
"operation=", "path=", "payload=",
                                                                    "payload_file="
"])
    except getopt.GetoptError as err:
        # print help information and exit:
        print str(err) # will print something like "option -a not recognized"
        usage()
        sys.exit(2)

    for o, a in opts:
        if o in ("-o", "--operation"):
            op = a
        elif o in ("-p", "--path"):
            path = a
        elif o in ("-P", "--payload"):
            payload = a
        elif o in ("-f", "--payload-file"):
            with open(a, 'r') as f:
                payload = f.read()
        elif o in ("-h", "--help"):
            usage()
            sys.exit()
        else:
            usage()
            sys.exit(2)

    if op is None:
        print "Operation must be specified"
        usage()
        sys.exit(2)

    if path is None:
        print "Path must be specified"
        usage()
        sys.exit(2)

    if not path.startswith("coap://"):
        print "Path must be conform to coap://host[:port]/path"
        usage()
        sys.exit(2)

    host, port, path = parse_uri(path)

    try:
        tmp = socket.gethostbyname(host)
        host = tmp
    except socket.gaierror:
        pass

```

```

client = HelperClient(server=(host, port))

if op == "GET":
if path is None:
    print "Path cannot be empty for a GET request"
    usage()
    sys.exit(2)

var = 1

try :
    while var == 1:
# Send the request GET to the smartplug to obtain the power
consumption
        response = client.get(path)
        power = response.payload

        # Parameters
        URL = 'http://localhost/emoncms/input/post'
        API_KEY = 'dad82a43676146f471444fbd5c7726c2' # Edit API_KEY
parameter according to emoncms data base
        node_name = str(host)
        start_time = time.time()
        time_between_requests = 10

        # Create payload
        emoncms_data = [('node', node_name), ('data', '{power:' +
str(power) + '}'), ('apikey', API_KEY),]

        # Post data on emoncms
        r = requests.post(URL , data = emoncms_data)

        # Wait 10 seconds
        time.sleep(time_between_requests)

        # Log in terminal
        for i in range(0,3):
            if emoncms_data[i][0] != 'data':
                print emoncms_data[i][0] + ' = ' +
emoncms_data[i][1]
            else:
                print 'power = ' + emoncms_data[i][1][7:-2] + '
Watts'

        print('\nElapsed time = ' + str(int(time.time() - start_time)) +
's\n')
        print('-----\n')

    except KeyboardInterrupt:
        client.stop()

elif op == "OBSERVE":
if path is None:
    print "Path cannot be empty for a GET request"
    usage()

```

```

        sys.exit(2)
    client.observe(path, client_callback_observe)

    elif op == "POST":
    if path is None:
        print "Path cannot be empty for a POST request"
        usage()
        sys.exit(2)
    if payload is None:
        print "Payload cannot be empty for a POST request"
        usage()
        sys.exit(2)
    print client
    response = client.post(path, payload)
    print response.pretty_print()
    client.stop()

    else:
    print "Operation not recognized"
    usage()
    sys.exit(2)

if __name__ == '__main__': # pragma: no cover
    main()

```

---

## COAPCLIENT\_HTTPSERVER.PY

```

from coapthon.client.helperclient import HelperClient
from coapthon.utils import parse_uri
import sys
import getopt
import socket
import sys
import requests
import time
import random
import BaseHTTPServer

# HTTP server address
HOST = '0.0.0.0'
PORT = 8081

class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self):
        # Respond to a GET request from HTTP Client.
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        message = "200 - Ok"
        self.wfile.write(bytes(message))

        # Get CoAP parameters from the path
        request_path = self.path
        request_path_split = request_path.split("/")

        host = request_path_split[1]
        port = int(request_path_split[2])

```



```

# Send CoAP POST to the smartplug
payload='0'
path = "pwr/rel"
client = HelperClient(server=(host, port))
response = client.post(path,payload)
client.stop()

# Print the state of the smartplug
print(host + ' switched')

# Delete log message of self.send_response(200)
def log_message(self, format, *args):
return

def run():
server_address = (HOST, PORT)
server_class = BaseHTTPServer.HTTPServer
httpd = server_class(server_address, MyHandler)
print time.asctime(), "Server Starts - %s:%s" % server_address
try:
httpd.serve_forever()
except KeyboardInterrupt:
pass
httpd.server_close()
print time.asctime(), "Server Stops - %s:%s" % server_address

run()

```